

(19)



(11)

EP 3 352 083 B1

(12)

EUROPEAN PATENT SPECIFICATION

(45) Date of publication and mention
of the grant of the patent:
06.11.2019 Bulletin 2019/45

(51) Int Cl.:
G06F 11/36 ^(2006.01) **G06F 9/445** ^(2018.01)
G06F 9/48 ^(2006.01)

(21) Application number: **16848168.7**

(86) International application number:
PCT/CN2016/099996

(22) Date of filing: **24.09.2016**

(87) International publication number:
WO 2017/050287 (30.03.2017 Gazette 2017/13)

(54) DEBUGGING METHOD, MULTI-CORE PROCESSOR, AND DEBUGGING EQUIPMENT

**FEHLERBESEITIGUNGSVERFAHREN, MEHRKERNPROZESSOR UND
FEHLERBESEITIGUNGSAUSRÜSTUNG**

PROCÉDÉ DE DÉBOGAGE, PROCESSEUR MULTI-COEURS ET ÉQUIPEMENT DE DÉBOGAGE

(84) Designated Contracting States:
**AL AT BE BG CH CY CZ DE DK EE ES FI FR GB
GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO
PL PT RO RS SE SI SK SM TR**

(30) Priority: **25.09.2015 CN 201510623830**

(43) Date of publication of application:
25.07.2018 Bulletin 2018/30

(73) Proprietor: **Huawei Technologies Co., Ltd.**
Longgang District
Shenzhen, Guangdong 518129 (CN)

(72) Inventors:
• **WANG, Mingfa**
Shenzhen
Guangdong 518129 (CN)

• **YU, Gang**
Shenzhen
Guangdong 518129 (CN)
• **WANG, Haichuan**
Shenzhen
Guangdong 518129 (CN)

(74) Representative: **Kreuz, Georg Maria**
Huawei Technologies Duesseldorf GmbH
Riesstraße 25
80992 München (DE)

(56) References cited:
WO-A1-2014/166526 CN-A- 1 779 652
CN-A- 102 346 708 CN-A- 103 729 288
CN-A- 105 224 454 US-A1- 2005 216 895
US-A1- 2007 266 376

EP 3 352 083 B1

Note: Within nine months of the publication of the mention of the grant of the European patent in the European Patent Bulletin, any person may give notice to the European Patent Office of opposition to that patent, in accordance with the Implementing Regulations. Notice of opposition shall not be deemed to have been filed until the opposition fee has been paid. (Art. 99(1) European Patent Convention).

Description

TECHNICAL FIELD

[0001] The present invention relates to the field of computer technologies, and in particular, to a debugging method, a multi-core processor, and a debugging device.

BACKGROUND

[0002] Program debugging is a process in which a compiled computer program is tested manually or by using a method such as a compiling program before being put into actual operation, so as to correct a syntax error and a logic error. This step is indispensable for ensuring correctness of a computer information system. Therefore, the compiled computer program needs to be loaded into a computer for debugging.

[0003] Generally, in the industry, code that runs in kernel space is referred to as kernel mode code, and code that runs in user space is referred to as user mode code. Currently, different debugging platforms need to be used to debug the kernel mode code and the user mode code. It should be noted that, a target machine described below refers to a physical host in which a debugged code runs, and a host machine described below refers to a physical host in which a debugger runs. For example, when user mode code is being debugged, a debugger needs to run in an operating system of a target machine to perform local debugging; or when user mode code is being debugged, a process tracing module (Ptrace) needs to be implanted into an operating system of a target machine, a debugging stub may further need to run in the operating system, and then a debugger runs in a host machine to perform remote debugging. When kernel mode code such as a kernel module is being debugged, a kernel debugger needs to be implanted into an operating system of a target machine, and then a debugger runs in a host machine to perform remote debugging.

[0004] CN102346708A discloses a debugger and a debugging method thereof. The debugging method comprises the following steps: after the debugging relation is established, a debugged task runs in a kernel mode or a user mode by means of the debugger; when the debugger receives a stop instruction sent by a top level or runs into an abnormal event during the debugging process of the debugged task or captures a concerned signal message during the debugging process of the debugged task in the user mode, the debugged task stops running, and a stop mark bit is set in the debugged task; and after the debugged task stops running, the debugger accesses the memory information and/or register information of the debugged task. by adopting the debugging method, the running status of the debugged task in user address space and kernel address space can be tracked and debugged.

SUMMARY

[0005] The present invention provides a debugging method, a multi-core processor, and a debugging device, so as to debug kernel mode code and user mode code on a same debugging platform.

[0006] According to a first aspect, the present invention provides a debugging method, applied to a target machine in a remote debugging system, where the target machine includes a multi-core processor, and the debugging method includes:

stopping running, by a core A of the multi-core processor after completing execution of a processing routine of a preset event, and sending a running stop signal to other cores in a process of stopping running, where the running stop signal is used to instruct the other cores to stop running, the other cores refer to all cores that are located in the multi-core processor and that belong to a same logical group as the core A, and the preset event is a kernel mode code processing function or a user mode code processing function;

after receiving a first stop termination instruction and resuming running, executing, by the core A, a debugging information collection function to collect debugging information of the preset event, and stopping running after completing the execution of the debugging information collection function, where the first stop termination instruction is sent by a debugging apparatus in the remote debugging system after the debugging apparatus determines that the core A stops running and that a reason that the core A stops running is a reason other than the preset event; after receiving a second stop termination instruction and resuming running, sending, by the core A, a running resumption instruction to the other cores, where the running resumption instruction is used to instruct the other cores to resume running, and the second stop termination instruction is sent by the debugging apparatus after the debugging apparatus sets a preset pending breakpoint as an operation object of the preset event according to the debugging information of the preset event; and

knocking, by the core A, the pending breakpoint in a process of running the operation object of the preset event, so as to enter a debugging state.

[0007] With reference to the first aspect, in a first implementation manner of the first aspect, the debugging information collection function includes a debugging interface function, a first debugging exception instruction is set in a header of the debugging interface function, and the first debugging exception instruction is used to instruct the core A to stop running; and the stopping running, by the core A after completing execution of a processing routine of a preset event, and sending a running stop signal to other cores in a process

of stopping running specifically includes:

after completing the execution of the processing routine of the preset event, starting, by the core A, to execute the first debugging exception instruction located in the header of the debugging interface function, and sending the running stop signal to the other cores in a process of executing the first debugging exception instruction.

[0008] With reference to the first implementation manner of the first aspect, in a second implementation manner of the first aspect, the debugging information collection function further includes a function associated with the preset event, a second debugging exception instruction is further set in a trailer of the debugging interface function, and the second debugging exception instruction is used to instruct the core A to stop running; and the executing, by the core A, a debugging information collection function to collect debugging information of the preset event, and stopping running after completing the execution of the debugging information collection function specifically includes:

executing, by the core A, the function associated with the preset event to collect the debugging information of the preset event, and then executing the second debugging exception instruction located in the trailer of the debugging interface function.

[0009] With reference to the second implementation manner of the first aspect, in a third implementation manner of the first aspect, the function associated with the preset event is located in a kernel debugging stub, and the kernel debugging stub is implanted into an operating system of the target machine in advance.

[0010] With reference to any one of the first aspect, or the first implementation manner of the first aspect to the third implementation manner of the first aspect, in a fourth implementation manner of the first aspect, the kernel mode code processing function is a kernel module loading function; and the user mode code processing function is a process creation function, a process switching function, a process blocking function, a process wakeup function, a thread creation function, a thread switching function, a thread blocking function, or a thread wakeup function.

[0011] With reference to the fourth implementation manner of the first aspect, in a fifth implementation manner of the first aspect, if the preset event is the kernel module loading function, the operation object of the preset event is a kernel module; if the preset event is the process creation function, the process switching function, the process blocking function, or the process wakeup function, the operation object of the preset event is a process; and if the preset event is the thread creation function, the thread switching function, the thread blocking function, or the thread wakeup function, the operation object of the preset event is a thread.

[0012] With reference to the fourth implementation manner of the first aspect or the fifth implementation manner of the first aspect, in a sixth implementation manner of the first aspect,

the debugging information of the preset event includes an identifier of the operation object of the preset event; and

if the preset event is the kernel module loading function, the process creation function, or the thread creation function, the debugging information of the preset event further includes a loading address of the operation object of the preset event.

[0013] With reference to any one of the first aspect, or the first implementation manner of the first aspect to the sixth implementation manner of the first aspect, in a seventh implementation manner of the first aspect, the multi-core processor is integrated into a system on chip SOC; the sending, by a core A, a running stop signal to other cores, where the running stop signal is used to instruct the other cores to stop running specifically includes:

sending, by the core A, the running stop signal to the other cores by using a hardware cross-trigger network of the SOC, where the running stop signal is used to instruct the other cores to stop running; and the sending, by the core A, a running resumption instruction to the other cores, where the running resumption instruction is used to instruct the other cores to resume running specifically includes:

sending, by the core A, the running resumption instruction to the other cores by using the hardware cross-trigger network of the SOC, where the running resumption instruction is used to instruct the other cores to resume running.

[0014] According to a second aspect, the present invention provides a debugging method, executed by a debugging apparatus deployed in a host machine, where the host machine is located in a remote debugging system, and the method includes:

determining, by the debugging apparatus after determining that a core A stops running, a reason that the core A stops running, where the core A belongs to a multi-core processor of a target machine in the remote debugging system;

if the reason that the core A stops running is a preset event, obtaining, by the debugging apparatus, debugging information of the preset event, and setting a preset pending breakpoint as an operation object of the preset event according to the debugging information of the preset event, where the preset event is a kernel mode code processing function or a user mode code processing function, and the operation object of the preset event is kernel mode code or user mode code; and

instructing, by the debugging apparatus, the core A to resume running, so that the core A knocks the

pending breakpoint in a process of running the operation object of the preset event, and the core A enters a debugging state.

[0015] With reference to the second aspect, in a first implementation manner of the second aspect, the method further includes:

if the reason that the core A stops running is a reason other than the preset event, instructing, by the debugging apparatus, the core A to resume running, to collect the debugging information of the preset event.

[0016] With reference to the second aspect or the first implementation manner of the second aspect, in a second implementation manner of the second aspect, the function of the kernel mode code is a kernel module loading function; and
the function of the user mode code is a process creation function, a process switching function, a process blocking function, a process wakeup function, a thread creation function, a thread switching function, a thread blocking function, or a thread wakeup function.

[0017] With reference to the second implementation manner of the second aspect, in a third implementation manner of the second aspect,

if the preset event is the kernel module loading function, the process creation function, or the thread creation function, the debugging information of the preset event includes a loading address of the operation object of the preset event; and

the obtaining, by the debugging apparatus, debugging information of the preset event, and setting a preset pending breakpoint as an operation object of the preset event according to the debugging information of the preset event specifically includes:

obtaining, by the debugging apparatus, the loading address of the operation object of the preset event, and setting the preset pending breakpoint as a destination address of the operation object of the preset event according to the loading address of the operation object of the preset event.

[0018] With reference to the second implementation manner of the second aspect, in a fourth implementation manner of the second aspect,

if the preset event is the process switching function, the process blocking function, the process wakeup function, the thread switching function, the thread blocking function, or the thread wakeup function, the debugging information of the preset event includes an identifier of the operation object of the preset event; and

the obtaining, by the debugging apparatus, debugging information of the preset event, and setting a preset pending breakpoint as an operation object of the preset event according to the debugging information of the preset event specifically includes:

obtaining, by the debugging apparatus, the identifier of the operation object of the preset event, determining a loading address of the operation object of the preset event according to the identifier of the operation object

of the preset event, and setting the preset pending breakpoint as a destination address of the operation object of the preset event according to the loading address of the operation object of the preset event.

[0019] With reference to the fourth implementation manner of the second aspect, in a fifth implementation manner of the second aspect,

a correspondence between the identifier of the operation object of the preset event and the loading address of the operation object of the preset event is pre-stored in storage space of the host machine; and

the determining a loading address of the operation object of the preset event according to the identifier of the operation object of the preset event specifically includes:

obtaining, from the storage space of the host machine, the loading address of the operation object of the preset event according to the identifier of the operation object of the preset event and the correspondence.

[0020] With reference to any one of the second aspect, or the first implementation manner of the second aspect to the fifth implementation manner of the second aspect, in a sixth implementation manner of the second aspect, the debugging apparatus includes a debugger and a debugging agent;

the determining, by the debugging apparatus after determining that a core A stops running, a reason that the core A stops running, and if the reason that the core A stops running is a preset event, obtaining, by the debugging apparatus, debugging information of the preset event, and setting a preset pending breakpoint as an operation object of the preset event according to the debugging information of the preset event specifically includes:

after determining, in a polling manner, that the core A stops running, determining, by the debugging agent, the reason that the core A stops running, and if the reason that the core A stops running is the preset event, reporting, by the debugging agent, a second message to the debugger, where the second message includes an identifier indicating that the core A stops running and an identifier of the preset event; and

after receiving the second message, obtaining, by the debugger, the debugging information of the preset event according to the identifier of the preset event, and setting the preset pending breakpoint as the operation object of the preset event according to the debugging information of the preset event; and the instructing, by the debugging apparatus, the core A to resume running specifically includes:

instructing, by the debugger, the debugging agent to send a second stop termination instruction to the core A, where the second stop termination instruction is used to instruct the core A to resume running.

[0021] With reference to the sixth implementation manner of the second aspect, in a seventh implementation manner of the second aspect,

the determining, by the debugging apparatus after determining that a core A stops running, a reason that the core A stops running, and if the reason that the core A stops running is a reason other than the preset event, instructing, by the debugging apparatus, the core A to resume running, to collect the debugging information of the preset event specifically includes:

after determining, in a polling manner, that the core A stops running, determining, by the debugging agent, the reason that the core A stops running, and if the reason that the core A stops running is a reason other than the preset event, sending, by the debugging agent, a first stop termination instruction to the core A, where the first stop termination instruction is used to instruct the core A to resume running, to collect the debugging information of the preset event.

[0022] With reference to any one of the second aspect, or the first implementation manner of the second aspect to the sixth implementation manner of the second aspect, in an eighth implementation manner of the second aspect, the debugging apparatus includes the debugger and the debugging agent; and the determining, by the debugging apparatus after determining that a core A stops running, a reason that the core A stops running, and if the reason that the core A stops running is a reason other than the preset event, instructing, by the debugging apparatus, the core A to resume running, to collect the debugging information of the preset event specifically includes:

after determining, in a polling manner, that the core A stops running, determining, by the debugging agent, the reason that the core A stops running, and if the reason that the core A stops running is a reason other than the preset event, reporting a first message to the debugger, where the first message includes the identifier indicating that the core A stops running and a default identifier; and after receiving the first message, instructing, by the debugger, the debugging agent to send a first stop termination instruction to the core A, where the first stop termination instruction is used to instruct the core A to resume running, to collect the debugging information of the preset event.

[0023] With reference to any one of the second aspect, or the first implementation manner of the second aspect to the eighth implementation manner of the second aspect, in a ninth implementation manner of the second aspect,

if the preset event is the kernel module loading function, the operation object of the preset event is a kernel module;

if the preset event is the process creation function, the process switching function, the process blocking function, or the process wakeup function, the operation object of the preset event is a process; and

if the preset event is the thread creation function, the

thread switching function, the thread blocking function, or the thread wakeup function, the operation object of the preset event is a thread.

[0024] According to a third aspect, the present invention provides a multi-core processor, applied to a target machine in a remote debugging system, where the multi-core processor includes a core A, other cores, and an input port, and the other cores refer to all cores that are located in the multi-core processor and that belong to a same logical group as the core A, where

the core A is configured to: stop running after completing execution of a processing routine of a preset event, and send a running stop signal to the other cores in a process of stopping running, where the running stop signal is used to instruct the other cores to stop running, and the preset event is a kernel mode code processing function or a user mode code processing function;

the input port is configured to: receive a first stop termination instruction, and forward the first stop termination instruction to the core A, where the first stop termination instruction is sent by a debugging apparatus in the remote debugging system after the debugging apparatus determines that the core A stops running and that a reason that the core A stops running is a reason other than the preset event;

the core A is further configured to: after receiving the first stop termination instruction and resuming running, execute a debugging information collection function to collect debugging information of the preset event, and stop running after completing the execution of the debugging information collection function;

the input port is further configured to: receive a second stop termination instruction, and forward the second stop termination instruction to the core A, where the second stop termination instruction is sent by the debugging apparatus after the debugging apparatus sets a preset pending breakpoint as an operation object of the preset event according to the debugging information of the preset event; and

the core A is further configured to: send a running resumption instruction to the other cores after receiving the second stop termination instruction and resuming running, where the running resumption instruction is used to instruct the other cores to resume running; and knock the pending breakpoint in a process of running the operation object of the preset event, so as to enter a debugging state.

[0025] With reference to the third aspect, in a first implementation manner of the third aspect,

the debugging information collection function includes a debugging interface function, a first debugging exception instruction is set in a header of the debugging interface function, and the first debugging exception instruction is used to instruct the core A to stop running; and

the core A is specifically configured to: after completing the execution of the processing routine of the preset event, start to execute the first debugging exception instruction located in the header of the debugging interface

function, and send the running stop signal to the other cores in a process of executing the first debugging exception instruction.

[0026] With reference to the first implementation manner of the third aspect, in a second implementation manner of the third aspect, the debugging information collection function further includes a function associated with the preset event, a second debugging exception instruction is further set in a trailer of the debugging interface function, and the second debugging exception instruction is used to instruct the core A to stop running; and the core A is specifically configured to: execute the function associated with the preset event to collect the debugging information of the preset event, and then execute the second debugging exception instruction located in the trailer of the debugging interface function.

[0027] With reference to the second implementation manner of the third aspect, in a third implementation manner of the third aspect, the function associated with the preset event is located in a kernel debugging stub, and the kernel debugging stub is implanted into an operating system of the target machine in advance.

[0028] With reference to any one of the third aspect, or the first implementation manner of the third aspect to the third implementation manner of the third aspect, in a fourth implementation manner of the third aspect, the kernel mode code processing function is a kernel module loading function; and the user mode code processing function is a process creation function, a process switching function, a process blocking function, a process wakeup function, a thread creation function, a thread switching function, a thread blocking function, or a thread wakeup function.

[0029] With reference to the fourth implementation manner of the third aspect, in a fifth implementation manner of the third aspect, if the preset event is the kernel module loading function, the operation object of the preset event is a kernel module; if the preset event is the process creation function, the process switching function, the process blocking function, or the process wakeup function, the operation object of the preset event is a process; and if the preset event is the thread creation function, the thread switching function, the thread blocking function, or the thread wakeup function, the operation object of the preset event is a thread.

[0030] With reference to the fourth implementation manner of the third aspect or the fifth implementation manner of the third aspect, in a sixth implementation manner of the third aspect, the debugging information of the preset event includes an identifier of the operation object of the preset event; and if the preset event is the kernel module loading function, the process creation function, or the thread creation func-

tion, the debugging information of the preset event further includes a loading address of the operation object of the preset event.

[0031] With reference to any one of the third aspect, or the first implementation manner of the third aspect to the sixth implementation manner of the third aspect, in a seventh implementation manner of the third aspect, the multi-core processor is integrated into a system on chip SOC;

the core A is specifically configured to send the running stop signal to the other cores by using a hardware cross-trigger network of the SOC, where the running stop signal is used to instruct the other cores to stop running; and the core A is specifically configured to send the running resumption instruction to the other cores by using the hardware cross-trigger network of the SOC, where the running resumption instruction is used to instruct the other cores to resume running.

[0032] According to a fourth aspect, the present invention provides a debugging apparatus, where the debugging device is deployed on a host machine in a remote debugging system, and the debugging device includes a debugging unit and a receiving unit, where the receiving unit is configured to: receive a message used for instructing a core A to stop running, and forward the message used for instructing the core A to stop running to the debugging unit, where a target machine located in the remote debugging system includes a multi-core processor, and the multi-core processor includes the core A;

the debugging unit is configured to: after determining that the core A stops running, determine a reason that the core A stops running;

if the reason that the core A stops running is a preset event, the debugging unit is further configured to: obtain debugging information of the preset event, and set a preset pending breakpoint as an operation object of the preset event according to the debugging information of the preset event, where the preset event is a kernel mode code processing function or a user mode code processing function, and the operation object of the preset event is kernel mode code or user mode code; and

the debugging unit is further configured to instruct the core A to resume running, so that the core A knocks the pending breakpoint in a process of running the operation object of the preset event, and the core A enters a debugging state.

[0033] With reference to the fourth aspect, in a first implementation manner of the fourth aspect, if the reason that the core A stops running is a reason other than the preset event, the debugging unit is further configured to instruct the core A to resume running, to collect the debugging information of the preset event.

[0034] With reference to the fourth aspect or the first implementation manner of the fourth aspect, in a second implementation manner of the fourth aspect, the function of the kernel mode code is a kernel module loading function; and

the function of the user mode code is a process creation function, a process switching function, a process blocking function, a process wakeup function, a thread creation function, a thread switching function, a thread blocking function, or a thread wakeup function.

[0035] With reference to the second implementation manner of the fourth aspect, in a third implementation manner of the fourth aspect, if the preset event is the kernel module loading function, the process creation function, or the thread creation function, the debugging information of the preset event includes a loading address of the operation object of the preset event; and the debugging unit is specifically configured to: obtain the loading address of the operation object of the preset event, and set the preset pending breakpoint as a destination address of the operation object of the preset event according to the loading address of the operation object of the preset event.

[0036] With reference to the second implementation manner of the fourth aspect, in a fourth implementation manner of the fourth aspect, if the preset event is the process switching function, the process blocking function, the process wakeup function, the thread switching function, the thread blocking function, or the thread wakeup function, the debugging information of the preset event includes an identifier of the operation object of the preset event; and the debugging unit is specifically configured to: obtain the identifier of the operation object of the preset event, determine a loading address of the operation object of the preset event according to the identifier of the operation object of the preset event, and set the preset pending breakpoint as a destination address of the operation object of the preset event according to the loading address of the operation object of the preset event.

[0037] With reference to the fourth implementation manner of the fourth aspect, in a fifth implementation manner of the fourth aspect, a correspondence between the identifier of the operation object of the preset event and the loading address of the operation object of the preset event is pre-stored in storage space of the host machine; and the debugging unit is specifically configured to obtain, from the storage space of the host machine, the loading address of the operation object of the preset event according to the identifier of the operation object of the preset event and the correspondence.

[0038] With reference to any one of the fourth aspect, or the first implementation manner of the fourth aspect to the fifth implementation manner of the fourth aspect, in a sixth implementation manner of the fourth aspect, the debugging unit includes a debugger and a debugging agent;

the debugging agent is specifically configured to: after determining that the core A stops running, determine the reason that the core A stops running, and if the reason that the core A stops running is the preset event, report

a second message to the debugger, where the second message includes an identifier indicating that the core A stops running and an identifier of the preset event; the debugger is specifically configured to: after receiving the second message, obtain the debugging information of the preset event according to the identifier of the preset event, and set the preset pending breakpoint as the operation object of the preset event according to the debugging information of the preset event; and the debugger is specifically configured to instruct the debugging agent to send a second stop termination instruction to the core A, where the second stop termination instruction is used to instruct the core A to resume running.

[0039] With reference to the sixth implementation manner of the fourth aspect, in a seventh implementation manner of the fourth aspect, the debugging agent is specifically configured to: after determining that the core A stops running, determine the reason that the core A stops running, and if the reason that the core A stops running is a reason other than the preset event, send a first stop termination instruction to the core A, where the first stop termination instruction is used to instruct the core A to resume running, to collect the debugging information of the preset event.

[0040] With reference to any one of the fourth aspect, or the first implementation manner of the fourth aspect to the sixth implementation manner of the fourth aspect, in an eighth implementation manner of the fourth aspect, the debugging apparatus further includes the debugger and the debugging agent; the debugging agent is specifically configured to: after determining that the core A stops running, determine the reason that the core A stops running, and if the reason that the core A stops running is a reason other than the preset event, report a first message to the debugger, where the first message includes the identifier indicating that the core A stops running and a default identifier; and the debugger is specifically configured to: after receiving the first message, instruct the debugging agent to send a first stop termination instruction to the core A, where the first stop termination instruction is used to instruct the core A to resume running, to collect the debugging information of the preset event.

[0041] It can be learned that according to the debugging method provided by the present invention, a debugging interface function is added to a trailer of a preset event that runs on a core A. The debugging interface function invokes a function that is associated with a type of the preset event and that is in a kernel debugging stub, to obtain debugging information of the preset event. A debugger sets a preset pending breakpoint as a destination address of an operation object of the preset event according to the debugging information of the preset event, so that the core A knocks the pending breakpoint in a subsequent process of running the operation object of the preset event, implementation manner of the fifth aspect,

if the preset event is the kernel module loading function, the process creation function, or the thread creation function, the debugging information of the preset event includes a loading address of the operation object of the preset event; and

the processor is specifically configured to: obtain the loading address of the operation object of the preset event, and set the preset pending breakpoint as a destination address of the operation object of the preset event according to the loading address of the operation object of the preset event.

[0042] With reference to the second implementation manner of the fifth aspect, in a fourth implementation manner of the fifth aspect,

if the preset event is the process switching function, the process blocking function, the process wakeup function, the thread switching function, the thread blocking function, or the thread wakeup function, the debugging information of the preset event includes an identifier of the operation object of the preset event; and

the processor is specifically configured to: obtain the identifier of the operation object of the preset event, determine a loading address of the operation object of the preset event according to the identifier of the operation object of the preset event, and set the preset pending breakpoint as a destination address of the operation object of the preset event according to the loading address of the operation object of the preset event.

[0043] With reference to the fourth implementation manner of the fifth aspect, in a fifth implementation manner of the fifth aspect,

a correspondence between the identifier of the operation object of the preset event and the loading address of the operation object of the preset event is pre-stored in storage space of the host machine; and

the processor is specifically configured to obtain, from the storage space of the host machine, the loading address of the operation object of the preset event according to the identifier of the operation object of the preset event and the correspondence.

[0044] It can be learned that according to the debugging method provided by the present invention, a debugging interface function is added to a trailer of a preset event that runs on a core A. The debugging interface function invokes a function that is associated with a type of the preset event and that is in a kernel debugging stub, to obtain debugging information of the preset event. A debugger sets a preset pending breakpoint as a destination address of an operation object of the preset event according to the debugging information of the preset event, so that the core A knocks the pending breakpoint in a subsequent process of running the operation object of the preset event, so as to enter a debugging state. In the present invention, the preset event is a kernel mode code processing function or a user mode code processing function. Therefore, according to the technical solutions provided by the present invention, kernel mode code and user mode code can be debugged on a same

debugging platform.

BRIEF DESCRIPTION OF DRAWINGS

[0045] To describe the technical solutions of the present invention more clearly, in the following the accompanying drawings are briefly introduced describing preferred embodiments of the present invention. Apparently, the accompanying drawings in the following description show merely some embodiments of the present invention.

FIG. 1 is a schematic structural diagram of a remote debugging system according to an embodiment of the present invention;

FIG. 2 is a schematic structural diagram of another remote debugging system according to an embodiment of the present invention;

FIG. 3 is a schematic flowchart of a debugging method according to an embodiment of the present invention;

FIG. 4A and FIG. 4B are a schematic flowchart of another debugging method according to an embodiment of the present invention;

FIG. 5 is a schematic structural diagram of a target machine according to an embodiment of the present invention;

FIG. 6.1 is a schematic structural diagram of a data header and a data body according to an embodiment of the present invention;

FIG. 6.2 is another schematic structural diagram of a data header and a data body according to an embodiment of the present invention;

FIG. 7 is a schematic structural diagram of a multi-core processor according to an embodiment of the present invention;

FIG. 8 is a schematic structural diagram of a debugging apparatus according to an embodiment of the present invention; and

FIG. 9 is a schematic structural diagram of a debugging device according to an embodiment of the present invention.

DESCRIPTION OF EMBODIMENTS

[0046] The following clearly and completely describes the technical solutions of the present invention with reference to the accompanying drawings showing preferred embodiments of the present invention.

Embodiment 1

[0047] This embodiment of the present invention provides a debugging method applied to a remote debugging system shown in FIG. 1. Referring to FIG. 1, the remote debugging system includes a target machine and a host machine, the target machine includes a multi-core processor, and a debugging apparatus is deployed on the

host machine. The target machine and the host machine are different physical hosts, and the target machine communicates with and connects to the host machine. It should be noted that, in the technical solution provided in this embodiment of the present invention, the multi-core processor may be a homogeneous multi-core processor, or may be a heterogeneous multi-core processor. If the target machine includes multiple multi-core processors, a core A may be any core of any multi-core processor in the target machine, instead of a specific core of a specific multi-core processor in the target machine.

[0048] The debugging method provided in this embodiment of the present invention is a cross debugging method, and is a system-level debugging method. Referring to FIG. 3, the debugging method provided in this embodiment of the present invention includes the following steps.

[0049] S101. The core A stops running after completing execution of a processing routine of a preset event, and sends a running stop signal to other cores in a process of stopping running, where the running stop signal is used to instruct the other cores to stop running, the other cores refer to all cores that are located in the multi-core processor and that belong to a same logical group as the core A, and the preset event is a kernel mode code processing function or a user mode code processing function.

[0050] As shown in FIG. 5, in this embodiment of the present invention, a debugging interface function is added to a trailer of the preset event. The debugging interface function is located outside the preset event, and is not a part of the preset event. A first debugging exception instruction is set in a header of the debugging interface function, and the core A stops running after completing execution of the first debugging exception instruction. A second debugging exception instruction is further set in a trailer of the debugging interface function, and the core A stops running after completing execution of the second debugging exception instruction.

[0051] Specifically, that the core A stops running after completing execution of a processing routine of a preset event, and sends a running stop signal to other cores in a process of stopping running includes: after completing the execution of the processing routine of the preset event, the core A starts to execute the first debugging exception instruction located in the header of the debugging interface function, and sends the running stop signal to the other cores in a process of executing the first debugging exception instruction. It should be noted that, the core A executes the first debugging exception instruction for a period of time. The core A first starts to execute the first debugging exception instruction, then sends the running stop signal to the other cores at a moment before the execution of the first debugging exception instruction is completed, and continues to execute the first debugging exception instruction after sending the running stop signal to the other cores. The core A stops running after completing the execution of the first debugging exception

instruction.

[0052] It should be noted that, all cores included in a same logical group are located in a same multi-core processor. Logical grouping means that multiple cores in a service association relationship are classified into a same logical group by a debugging engineer according to experience. As shown in FIG. 1, the core A, a core B, a core C, and a core D are located in a same logical group. In this case, that the core A sends a running stop signal to other cores is specifically: the core A sends the running stop signal to the core B, the core C, and the core D.

[0053] In an embodiment of the present invention, if the target machine includes a system on chip SOC, and the multi-core processor including the core A is integrated into the SOC, that the core A sends a running stop signal to other cores, where the running stop signal is used to instruct the other cores to stop running is specifically: the core A sends the running stop signal to the other cores by using a hardware cross-trigger network of the SOC, where the running stop signal is used to instruct the other cores to stop running. It should be noted that, when sending a running resumption instruction to the other cores by using the hardware cross-trigger network of the SOC, the core A sends the running resumption instruction in a broadcast form. An advantage of transmitting a signal by using the hardware cross-trigger network of the system on chip SOC lies in: compared with transmitting a signal by using software, transmitting a signal by using hardware can improve time validity of transmission of the running stop signal.

[0054] It should be noted that, generally, in the industry, code that runs in kernel space is referred to as kernel mode code, and code that runs in user space is referred to as user mode code. The kernel mode code processing function is a kernel module loading function; and the user mode code processing function is a process creation function, a process switching function, a process blocking function, a process wakeup function, a thread creation function, a thread switching function, a thread blocking function, or a thread wakeup function. It is easily learned that regardless of multiple processes or multiple threads that run on a heterogeneous multi-core processor in an asymmetric multi-processing (AMP, asymmetric multi-processing) structure, or multiple processes or multiple threads that run on a homogeneous multi-core processor in a symmetric multi-processing (SMP, symmetric multi-processing) structure, synchronous debugging can be implemented by using the technical solution provided in this embodiment of the present invention.

[0055] S102. After determining that the core A stops running, the debugging apparatus determines a reason that the core A stops running, and if the reason that the core A stops running is a reason other than the preset event, the debugging apparatus sends a first stop termination instruction to the core A, where the first stop termination instruction is used to instruct the core A to resume running.

[0056] It should be noted that, in step S101, the core

A stops running after completing the execution of the processing routine of the preset event, and instructs the other cores to stop running; in step S102, the debugging apparatus sends the first stop termination instruction to the core A. Such a solution is designed to implement synchronous debugging. Referring to FIG. 1, the core A, the core B, the core C, and the core D are located in the same logical group. Data exchange is performed between code that runs on at least one core (such as the core B) of the core B, the core C, or the core D and code that runs on the core A. In this case, during debugging of the core A, if the core B continues to run, distortion may occur because data exchange cannot be correctly performed between the code that runs on the core B and the code that runs on the core A. Therefore, in the solution provided in this embodiment of the present invention, during debugging of the core A, the other cores that belong to the same logical group as the core A stop running, to avoid distortion. Further, the core A and the other cores first synchronously stop running, and then the core A resumes running for debugging. In addition, in order that the core A and the other cores synchronously stop running, a time at which the running stop signal sent by the core A is transmitted to the other cores and a time at which the core A stops running are further fully considered in design of the solution. Specifically, after stopping running, the core A sends a running stop instruction to the other cores in a running stop period, so that the core A and the other cores synchronously stop running, thereby implementing relatively precise synchronous debugging.

[0057] It should be noted that, there is preset storage space in a memory of the target machine, the preset storage space is specified by a debugging engineer in the memory of the target machine in advance, and both a size and an address of the preset storage space are fixed. The preset storage space is used to store a data header. Initially, the data header includes first default information and an address of a data body, and certainly, the first default information may be empty. The data body includes second default information, and certainly, the second default information may also be empty.

[0058] As shown in FIG. 6.1, if the core A stops running after completing the execution of the operation routine of the preset event, the data header still includes the first default information and a location of the data body, and content of the data body is the second default information. Correspondingly, if the core A stops running after completing the execution of the operation routine of the preset event, that after determining that the core A stops running, the debugging apparatus determines a reason that the core A stops running in S102 specifically includes: after determining, in a polling manner, that the core A stops running, the debugging apparatus determines, by reading content of the data header, the reason that the

core A stops running is a reason other than the preset event.

[0059] Referring to FIG. 4A and FIG. 4B, in an embodiment of the present invention, the debugging apparatus includes a debugging agent and a debugger, and the debugging apparatus is applied to the remote debugging system described in FIG. 2. It should be noted that, the debugging agent and the debugger may be located in a same host machine, or may be located in different host machines. For example, in FIG. 2, the debugging agent is located in a first host machine, and the debugger is located in a second host machine. As shown in FIG. 4A, if the debugging apparatus includes the debugging agent and the debugger, S102 in FIG. 3 specifically includes the following steps.

[0060] S202. After determining, in a polling manner, that the core A stops running, the debugging agent determines the reason that the core A stops running, and if the reason that the core A stops running is a reason other than the preset event, the debugging agent reports a first message to the debugger, where the first message includes an identifier indicating that the core A stops running and a default identifier.

[0061] With reference to content explained in detail above, if the core A stops running after completing the execution of the operation routine of the preset event, it can be learned that S202 specifically includes: after determining, in a polling manner, that the core A stops running, the debugging agent determines, by reading the content of the data header, the reason that the core A stops running. Because the content of the data header includes the first default information, the debugging agent can determine, according to the first default information, that the reason that the core A stops running is a reason other than the preset event, and report, to the debugger, the first message including the identifier indicating that the core A stops running and the default identifier.

[0062] S203. After receiving the first message, the debugger instructs the debugging agent to send the first stop termination instruction to the core A, where the first stop termination instruction is used to instruct the core A to resume running.

[0063] S204. The debugging agent sends the first stop termination instruction to the core A.

[0064] In an embodiment of the present invention, if the debugging apparatus includes the debugging agent and the debugger, S102 in FIG. 3 specifically includes: after determining, in a polling manner, that the core A stops running, the debugging agent determines the reason that the core A stops running, and if the reason that the core A stops running is a reason other than the preset event, the debugging agent sends the first stop termination instruction to the core A, where the first stop termination instruction is used to instruct the core A to resume running.

[0065] For details, refer to the foregoing explanation of step S402. Details are not described herein again. It should be noted that, in this embodiment of the present

invention, the debugging agent no longer reports, to the debugger, the first message including the identifier indicating that the core A stops running and the default identifier. Instead, the debugging agent sends the first stop termination instruction to the core A after reading the first default information from the data header and determining that the reason that the core A stops running is a reason other than the preset event. A difference between this embodiment and the foregoing embodiment lies in: the debugging agent sends the first stop termination instruction to the core A because the debugging agent determines that the reason that the core A stops running is a reason other than the preset event, instead of receiving an instruction of the debugger. Compared with the foregoing embodiment, the debugging agent shares more work of the debugger in this embodiment. Therefore, fewer function requirements are imposed on the debugger, which helps to simplify the debugger.

[0066] S103. After receiving the first stop termination instruction and resuming running, the core A executes a debugging information collection function to collect debugging information of the preset event, and stops running after completing the execution of the debugging information collection function.

[0067] Referring to FIG. 5, it should be noted that, the debugging information collection function includes the debugging interface function added to the trailer of the preset event and a function associated with the preset event. The function associated with the preset event is located in a kernel debugging stub, and the kernel debugging stub is implanted into an operating system of the target machine in advance.

[0068] As described for S101 in detail above, the first debugging exception instruction is set in the header of the debugging interface function, and the second debugging exception instruction is set in the trailer of the debugging interface function. In a process of executing the debugging interface function, the function that is associated with the preset event and that is located in the kernel debugging stub is invoked.

[0069] Specifically, S103 includes: the core A executes the function associated with the preset event to collect the debugging information of the preset event, and then executes the second debugging exception instruction located in the trailer of the debugging interface function. The core A stops running after the core A completes the execution of the second debugging exception instruction.

[0070] It should be noted that, the debugging information of the preset event includes an identifier of the preset event and an identifier of an operation object of the preset event. If the preset event is the kernel module loading function, the process creation function, or the thread creation function, the debugging information of the preset event further includes a loading address of the operation object of the preset event. It can be learned that an information amount of the debugging information of the preset event is relatively small, and is generally only about 100 rows. Therefore, according to the technical

solution provided in this embodiment of the present invention, an amount of data to be processed by a processor of the target machine can be reduced, thereby improving debugging efficiency.

[0071] S104. After determining that the core A stops running, the debugging apparatus determines the reason that the core A stops running, and if the reason that the core A stops running is the preset event, the debugging apparatus obtains the debugging information of the preset event, and sets a preset pending breakpoint as an operation object of the preset event according to the debugging information of the preset event, where the operation object of the preset event is the kernel mode code or the user mode code.

[0072] The debugging apparatus sends a second stop termination instruction to the core A, where the second stop termination instruction is used to instruct the core A to resume running.

[0073] As shown in FIG. 6.2, it should be noted that, if the core A stops running after completing the execution of the debugging information collection function, in a process of executing the debugging information collection function, the core A writes, into the data header to replace default information, an identifier that is of the preset event and that is used for triggering the core A to execute the debugging information collection function, and writes the identifier of the operation object of the preset event into the data body. If the preset event is the kernel module loading function, the process creation function, or the thread creation function, the core A further writes the loading address of the operation object of the preset event into the data body. It should be noted that, the loading address of the operation object of the preset event may be continuous or discontinuous. It is easily understood that, if the preset event is the process switching function, the process blocking function, the process wakeup function, the thread switching function, the thread blocking function, or the thread wakeup function, the debugging information collection function includes the identifier the preset event and the identifier of the operation object of the preset event. If the preset event is the kernel module loading function, the process creation function, or the thread creation function, the debugging information collection function includes the identifier of the preset event, the identifier of the operation object of the preset event, and the loading address of the operation object of the preset event.

[0074] Correspondingly, if the core A stops running after completing the execution of the debugging information collection function, that after determining that the core A stops running, the debugging apparatus determines the reason that the core A stops running in S104 specifically includes: after determining, in a polling manner, that the core A stops running, the debugging apparatus determines, by reading the content of the data header, the reason that the core A stops running. Because the content of the data header includes the identifier of the preset event, the debugging apparatus can

determine, according to the identifier of the preset event, that the reason that the core A stops running is the preset event.

[0075] It should be noted that, that the debugging apparatus obtains the debugging information of the preset event, and sets a preset pending breakpoint as an operation object of the preset event according to the debugging information of the preset event in S104 specifically includes:

if the debugging apparatus determines, according to the identifier of the preset event, that the preset event is the kernel module loading function, the process creation function, or the thread creation function, the debugging apparatus reads, from the data body, the loading address of the operation object of the preset event according to the address of the data body located in the data header, and then sets the preset pending breakpoint as a destination address of the operation object of the preset event according to the loading address of the operation object of the preset event. Certainly, the destination address of the operation object of the preset event is located in the loading address of the operation object of the preset event.

[0076] Alternatively, if the debugging apparatus determines, according to the identifier of the preset event, that the preset event is the process switching function, the process blocking function, the process wakeup function, the thread creation function, the thread switching function, or the thread blocking function, the debugging apparatus reads, from the data body, the identifier of the operation object of the preset event according to the address of the data body located in the data header, determines the loading address of the operation object of the preset event according to the identifier of the operation object of the preset event, and sets the preset pending breakpoint as a destination address of the operation object of the preset event according to the loading address of the operation object of the preset event.

[0077] It should be noted that, because a correspondence between the identifier of the operation object of the preset event and the loading address of the operation object of the preset event is pre-stored in storage space of the second host machine, the debugging apparatus determines the loading address of the operation object of the preset event according to the identifier of the operation object of the preset event. Specifically, the debugging apparatus obtains, from the storage space of the second host machine, the loading address of the operation object of the preset event according to the identifier of the operation object of the preset event and the correspondence.

[0078] With reference to FIG. 6.1 and FIG. 6.2, it can be learned that in the technical solution provided in this embodiment of the present invention, if the preset event is the kernel module loading function, the process creation function, the process switching function, the process blocking function, the process wakeup function, the thread creation function, the thread switching function,

the thread blocking function, or the thread wakeup function, and if the core A stops running after completing the execution of the debugging information collection function, content of the data header and the data body is shown in FIG. 6.2; if the core A stops running due to another event, content of the data header and the data body is shown in FIG. 6.1. Certainly, a case in which the execution of the debugging information collection function is completed is excluded from the another event. That is, the data header and the data body include only one message. When obtaining information from the data header, the debugging apparatus does not need to distinguish which is target information. In addition, because the data header and the data body include only one message, the size of the preset storage space in the memory of the target machine is fixed, and space occupied by the preset storage space is extremely small, thereby avoiding occupying excessive storage resources of the target machine, and improving efficiency of obtaining information.

[0079] It should be noted that, if the preset event is the kernel module loading function, the operation object of the preset event is a kernel module; if the preset event is the process creation function, the process switching function, the process blocking function, or the process wakeup function, the operation object of the preset event is a process; and if the preset event is the thread creation function, the thread switching function, the thread blocking function, or the thread wakeup function, the operation object of the preset event is a thread. The kernel module may be a dynamic patch, a hot patch, a dynamic library, or the like.

[0080] Referring to FIG. 4A and FIG. 4B, in an embodiment of the present invention, the debugging apparatus includes a debugging agent and a debugger, and the debugging apparatus is applied to the remote debugging system described in FIG. 2. It should be noted that, the debugging agent and the debugger may be located in a same host machine, or may be located in different host machines. For example, in FIG. 2, the debugging agent is located in a first host machine, and the debugger is located in a second host machine. As shown in FIG. 4B, if the debugging apparatus includes the debugging agent and the debugger, S104 in FIG. 3 specifically includes the following steps.

[0081] S206. After determining that the core A stops running, the debugging agent determines the reason that the core A stops running, and if the reason that the core A stops running is the preset event, the debugging agent reports a second message to the debugger, where the second message includes the identifier indicating that the core A stops running and an identifier of the preset event.

[0082] It should be noted that, if the core A stops running after completing the execution of the debugging information collection function, S206 specifically includes: after determining, in a polling manner, that the core A stops running, the debugging agent determines, by read-

ing the content of the data header, the reason that the core A stops running. Because the content of the data header includes the identifier of the preset event, the debugging agent can determine, according to the identifier of the preset event, that the reason that the core A stops running is the preset event, and report, to the debugger, the second message including the identifier indicating that the core A stops running and the identifier of the preset event.

[0083] S207. After receiving the second message, the debugger obtains the debugging information of the preset event according to the identifier of the preset event, and sets the preset pending breakpoint as the operation object of the preset event according to the debugging information of the preset event.

[0084] The debugger instructs the debugging agent to send the second stop termination instruction to the core A, where the second stop termination instruction is used to instruct the core A to resume running.

[0085] It should be noted that, that the debugger obtains the debugging information of the preset event according to the identifier of the preset event specifically includes: the debugger reads, from the data body, the identifier of the operation object of the preset event or the loading address of the operation object of the preset event according to the identifier of the preset event.

[0086] In an implementation manner of the present invention, if it is determined, according to the identifier of the preset event, that the preset event is the kernel module loading function, the process creation function, or the thread creation function, the debugger reads, from the data body, the loading address of the operation object of the preset event according to the address of the data body located in the data header, and then sets the preset pending breakpoint as the destination address of the operation object of the preset event according to the loading address of the operation object of the preset event. Certainly, the destination address of the operation object of the preset event is located in the loading address of the operation object of the preset event.

[0087] In another implementation manner of the present invention, if it is determined, according to the identifier of the preset event, that the preset event is the process switching function, the process blocking function, the process wakeup function, the thread creation function, the thread switching function, or the thread blocking function, the debugger reads, from the data body, the identifier of the operation object of the preset event according to the address of the data body located in the data header, determines the loading address of the operation object of the preset event according to the identifier of the operation object of the preset event, and sets the preset pending breakpoint as the destination address of the operation object of the preset event according to the loading address of the operation object of the preset event.

[0088] It should be noted that, because the correspondence between the identifier of the operation object

of the preset event and the loading address of the operation object of the preset event is pre-stored in the storage space of the second host machine, the debugger determines the loading address of the operation object of the preset event according to the identifier of the operation object of the preset event. Specifically, the debugger obtains, from the storage space of the second host machine, the loading address of the operation object of the preset event according to the identifier of the operation object of the preset event and the correspondence.

[0089] It should be noted that, the loading address of the operation object of the preset event in this embodiment of the present invention refers to a loading address of a target code of the operation object of the preset event, and the loading address of the target code of the operation object of the preset event refers to a loading address, on the target machine side, of the target code of the operation object of the preset event. A loading address of a source code of the operation object of the preset event refers to a loading address, on the second host machine side, of the source code of the operation object of the preset event. In this case, specifically, that the debugger sets the preset pending breakpoint as the destination address of the operation object of the preset event according to the loading address of the operation object of the preset event includes:

the debugger creates a symbol table according to a correspondence between the loading address of the target code of the operation object of the preset event and the loading address of the source code of the operation object of the preset event; and then the debugger determines, according to the symbol table, a target address that is on the target code side of the operation object of the preset event and that is mapped to the pending breakpoint (or the pending breakpoint), and sets the pending breakpoint as the destination address of the operation object of the preset event. The pending breakpoint (or the pending breakpoint) is set by a debugging engineer on the source code side of the operation object of the preset event in advance.

[0090] In this embodiment of the present invention, that "the debugger instructs the debugging agent to send the second stop termination instruction" occurs after "the debugger sets the preset pending breakpoint as the operation object of the preset event". In another embodiment of the present invention, that "the debugger instructs the debugging agent to send the second stop termination instruction" may occur at any moment after "the debugger obtains the debugging information of the preset event" and before "the debugger sets the preset pending breakpoint as the operation object of the preset event".

[0091] S208. The debugging agent sends a second stop termination instruction to the core A.

[0092] S105. The core A sends a running resumption instruction to the other cores after receiving the second stop termination instruction and resuming running, where the running resumption instruction is used to instruct the

other cores to resume running.

[0093] The core A knocks the pending breakpoint in a process of running the operation object of the preset event, so as to enter a debugging state.

[0094] As explained for step S301 above, if the target machine includes the system on chip SOC, and the multi-core processor including the core A is integrated into the SOC, that the core A sends a running resumption instruction to the other cores, where the running resumption instruction is used to instruct the other cores to resume running is specifically: the core A sends the running resumption instruction to the other cores by using the hardware cross-trigger network of the SOC, where the running resumption instruction is used to instruct the other cores to resume running. It should be noted that, when sending the running resumption instruction to the other cores by using the hardware cross-trigger network of the SOC, the core A sends the running resumption instruction in a broadcast form. Similarly, an advantage of transmitting a signal by using the hardware cross-trigger network of the SOC lies in that time validity of transmission of the running resumption instruction can be improved.

[0095] It can be learned from above that, according to the debugging method provided in this embodiment of the present invention, a debugging interface function is added to a trailer of a preset event that runs on a core A. The debugging interface function invokes a function that is associated with a type of the preset event and that is in a kernel debugging stub, to obtain debugging information of the preset event. A debugger may set a preset pending breakpoint as a destination address of an operation object of the preset event according to the debugging information of the preset event, so that the core A knocks the pending breakpoint in a subsequent process of running the operation object of the preset event, so as to enter a debugging state. In this embodiment of the present invention, the preset event is a kernel mode code processing function or a user mode code processing function. Therefore, according to the technical solution provided in this embodiment of the present invention, kernel mode code and user mode code can be debugged on a same debugging platform.

Embodiment 2

[0096] Referring to FIG. 7, this embodiment of the present invention provides a multi-core processor applied to the debugging method described in Embodiment 1, and the multi-core processor is deployed on a target machine in the remote debugging system shown in FIG. 1. As shown in FIG. 7, the multi-core processor includes a core A, an input port, and other cores. The other cores include a core B, a core C, and a core D. The core B, the core C, the core D, and the core A all are located in the multi-core processor and belong to a same logical group.

[0097] The core A is configured to: stop running after completing execution of a processing routine of a preset event, and send a running stop signal to the other cores

in a process of stopping running, where the running stop signal is used to instruct the other cores to stop running, and the preset event is a kernel mode code processing function or a user mode code processing function.

[0098] As shown in FIG. 5, in this embodiment of the present invention, a debugging interface function is added to a trailer of the preset event. The debugging interface function is located outside the preset event, and is not a part of the preset event. A first debugging exception instruction is set in a header of the debugging interface function, and the core A stops running after completing execution of the first debugging exception instruction. Therefore, the core A is specifically configured to: after completing the execution of the processing routine of the preset event, start to execute the first debugging exception instruction located in the header of the debugging interface function, and send the running stop signal to the other cores in a process of executing the first debugging exception instruction.

[0099] In an embodiment of the present invention, if the target machine includes a system on chip SOC, and the multi-core processor including the core A is integrated into the SOC, the core A is specifically configured to send the running stop signal to the other cores by using a hardware cross-trigger network of the SOC. The running stop signal is used to instruct the other cores to stop running. It should be noted that, when sending a running resumption instruction to the other cores by using the hardware cross-trigger network of the SOC, the core A sends the running resumption instruction in a broadcast form. An advantage of transmitting a signal by using the hardware cross-trigger network of the system on chip SOC lies in that time validity of transmission of the running stop signal is improved.

[0100] It should be noted that, the kernel mode code processing function is a kernel module loading function; and the user mode code processing function is a process creation function, a process switching function, a process blocking function, a process wakeup function, a thread creation function, a thread switching function, a thread blocking function, or a thread wakeup function. It is easily learned that regardless of multiple processes or multiple threads that run on a heterogeneous multi-core processor in an asymmetric multi-processing (AMP, asymmetric multi-processing) structure, or multiple processes or multiple threads that run on a homogeneous multi-core processor in a symmetric multi-processing (SMP, symmetric multi-processing) structure, synchronous debugging can be implemented by using the multi-core processor provided in this embodiment of the present invention.

[0101] Referring to FIG. 7, the core A, the core B, the core C, and the core D are located in the same logical group. Data exchange is performed between code that runs on at least one core (such as the core B) of the core B, the core C, or the core D and code that runs on the core A. In this case, during debugging of the core A, if the core B continues to run, distortion may occur because data exchange cannot be correctly performed between

the code that runs on the core B and the code that runs on the core A. Therefore, in the solution provided in this embodiment of the present invention, during debugging of the core A, the other cores that belong to the same logical group as the core A stop running, to avoid distortion. Further, the core A and the other cores first synchronously stop running, and then the core A resumes running for debugging. In addition, in order that the core A and the other cores synchronously stop running, a time at which the running stop signal sent by the core A is transmitted to the other cores and a time at which the core A stops running are further fully considered in design of the solution. Specifically, after stopping running, the core A sends a running stop instruction to the other cores in a running stop period, so that the core A and the other cores synchronously stop running, thereby implementing relatively precise synchronous debugging.

[0102] The input port is configured to: receive a first stop termination instruction, and forward the first stop termination instruction to the core A, where the first stop termination instruction is sent by a debugging apparatus in the remote debugging system after the debugging apparatus determines that the core A stops running and that a reason that the core A stops running is a reason other than the preset event.

[0103] The core A is further configured to: after receiving the first stop termination instruction and resuming running, execute a debugging information collection function to collect debugging information of the preset event, and stop running after completing the execution of the debugging information collection function.

[0104] Referring to FIG. 5, it should be noted that, the debugging information collection function includes the debugging interface function added to the trailer of the preset event and a function associated with the preset event. The function associated with the preset event is located in a kernel debugging stub, and the kernel debugging stub is implanted into an operating system of the target machine in advance.

[0105] A second debugging exception instruction is set in a trailer of the debugging interface function. In a process of executing the debugging interface function, the function that is associated with the preset event and that is located in the kernel debugging stub is invoked.

[0106] It should be noted that, the core A is specifically configured to: execute the function associated with the preset event to collect the debugging information of the preset event, and then execute the second debugging exception instruction located in the trailer of the debugging interface function. The core A stops running after the core A completes the execution of the second debugging exception instruction.

[0107] It should be noted that, the debugging information of the preset event includes an identifier of the preset event and an identifier of an operation object of the preset event. If the preset event is the kernel module loading function, the process creation function, or the thread creation function, the debugging information of the preset

event further includes a loading address of the operation object of the preset event. It can be learned that an information amount of the debugging information of the preset event is relatively small, and is generally only about 100 rows. Therefore, according to the technical solution provided in this embodiment of the present invention, an amount of data to be processed by a processor of the target machine can be reduced, thereby improving debugging efficiency.

[0108] The input port is further configured to: receive a second stop termination instruction, and forward the second stop termination instruction to the core A, where the second stop termination instruction is sent by the debugging apparatus after the debugging apparatus sets a preset pending breakpoint as the operation object of the preset event according to the debugging information of the preset event.

[0109] It should be noted that, if the preset event is the kernel module loading function, the operation object of the preset event is a kernel module; if the preset event is the process creation function, the process switching function, the process blocking function, or the process wakeup function, the operation object of the preset event is a process; and if the preset event is the thread creation function, the thread switching function, the thread blocking function, or the thread wakeup function, the operation object of the preset event is a thread. The kernel module may be a dynamic patch, a hot patch, a dynamic library, or the like.

[0110] The core A is further configured to: send a running resumption instruction to the other cores after receiving the second stop termination instruction and resuming running, where the running resumption instruction is used to instruct the other cores to resume running; and knock the pending breakpoint in a process of running the operation object of the preset event, so as to enter a debugging state.

[0111] As described above, if the target machine includes the system on chip SOC, and the multi-core processor including the core A is integrated into the SOC, the core A is specifically configured to send the running resumption instruction to the other cores by using the hardware cross-trigger network of the SOC. The running resumption instruction is used to instruct the other cores to resume running. It should be noted that, when sending the running resumption instruction to the other cores by using the hardware cross-trigger network of the SOC, the core A sends the running resumption instruction in a broadcast form. Similarly, an advantage of transmitting a signal by using the hardware cross-trigger network of the SOC lies in that time validity of transmission of the running resumption instruction can be improved.

[0112] It can be learned from above that, according to the multi-core processor provided in this embodiment of the present invention, a debugging interface function is added to a trailer of a preset event that runs on a core A. The debugging interface function invokes a function that is associated with a type of the preset event and that

is in a kernel debugging stub, to obtain debugging information of the preset event. Therefore, a debugging apparatus sets a preset pending breakpoint as a destination address of an operation object of the preset event according to the debugging information of the preset event, so that the core A knocks the pending breakpoint in a subsequent process of running the operation object of the preset event, so as to enter a debugging state. In this embodiment of the present invention, the preset event is a kernel mode code processing function or a user mode code processing function. Therefore, according to the multi-core processor provided in this embodiment of the present invention, kernel mode code and user mode code can be debugged on a same debugging platform.

Embodiment 3

[0113] Referring to FIG. 8, this embodiment of the present invention includes a debugging apparatus applied to the debugging method described in Embodiment 1, and the debugging apparatus is deployed on a host machine in the remote debugging system shown in FIG. 1. As shown in FIG. 8, the debugging apparatus includes a debugging unit and a receiving unit.

[0114] The receiving unit is configured to: receive a message used for instructing a core A to stop running, and forward the message used for instructing the core A to stop running to the debugging apparatus, where a target machine located in the remote debugging system includes a multi-core processor, and the multi-core processor includes the core A.

[0115] The debugging unit is configured to: after determining that the core A stops running, determine a reason that the core A stops running.

[0116] If the reason that the core A stops running is a preset event, the debugging unit is further configured to: obtain debugging information of the preset event, and set a preset pending breakpoint as an operation object of the preset event according to the debugging information of the preset event, where the preset event is a kernel mode code processing function or a user mode code processing function, and the operation object of the preset event is the kernel mode code or the user mode code.

[0117] It should be noted that, the kernel mode code processing function is a kernel module loading function; and the user mode code processing function is a process creation function, a process switching function, a process blocking function, a process wakeup function, a thread creation function, a thread switching function, a thread blocking function, or a thread wakeup function. If the preset event is the kernel module loading function, the operation object of the preset event is a kernel module; if the preset event is the process creation function, the process switching function, the process blocking function, or the process wakeup function, the operation object of the preset event is a process; and if the preset event is the thread creation function, the thread switching function, the thread blocking function, or the thread wakeup

function, the operation object of the preset event is a thread. The kernel module may be a dynamic patch, a hot patch, a dynamic library, or the like.

[0118] It should be noted that, there is preset storage space in a memory of the target machine, the preset storage space is specified by a debugging engineer in the memory of the target machine in advance, and both a size and an address of the preset storage space are fixed. The preset storage space is used to store a data header. Initially, the data header includes first default information and an address of a data body, and certainly, the first default information may be empty. The data body includes second default information, and certainly, the second default information may also be empty.

[0119] As shown in FIG. 6.2, if the core A stops running after completing execution of the debugging information collection function, in a process of executing the debugging information collection function, the core A writes, into the data header to replace default information, an identifier that is of the preset event and that is used for triggering the core A to execute the debugging information collection function, and writes an identifier of the operation object of the preset event into the data body. If the preset event is the kernel module loading function, the process creation function, or the thread creation function, the core A further writes a loading address of the operation object of the preset event into the data body. It should be noted that, the loading address of the operation object of the preset event may be continuous or discontinuous.

[0120] If the reason that the core A stops running is the preset event, the debugging unit is specifically configured to: after determining that the core A stops running, determine, by reading content of the data header, the reason that the core A stops running. Because the content of the data header includes the identifier of the preset event, the debugging unit can determine, according to the identifier of the preset event, that the reason that the core A stops running is the preset event.

[0121] Further, if it is determined, according to the identifier of the preset event, that the preset event is the kernel module loading function, the process creation function, or the thread creation function, the debugging unit is specifically configured to: read, from the data body, the loading address of the operation object of the preset event according to the address of the data body located in the data header, and then set the preset pending breakpoint as a destination address of the operation object of the preset event according to the loading address of the operation object of the preset event.

[0122] Alternatively, if it is determined, according to the identifier of the preset event, that the preset event is the process switching function, the process blocking function, the process wakeup function, the thread creation function, the thread switching function, or the thread blocking function, the debugging unit is specifically configured to: read, from the data body, the identifier of the operation object of the preset event according to the ad-

dress of the data body located in the data header, determine the loading address of the operation object of the preset event according to the identifier of the operation object of the preset event, and set the preset pending breakpoint as a destination address of the operation object of the preset event according to the loading address of the operation object of the preset event.

[0123] It should be noted that, a correspondence between the identifier of the operation object of the preset event and the loading address of the operation object of the preset event is pre-stored in storage space of the second host machine. Therefore, the debugging unit is specifically configured to obtain, from the storage space of the second host machine, the loading address of the operation object of the preset event according to the identifier of the operation object of the preset event and the correspondence.

[0124] It should be noted that, if the reason that the core A stops running is a reason other than the preset event, the debugging unit in the debugging apparatus provided in this embodiment of the present invention is further configured to instruct the core A to resume running, to collect the debugging information of the preset event.

[0125] Specifically, if the reason that the core A stops running is a reason other than the preset event, the debugging unit is specifically configured to: after determining that the core A stops running, determine, by reading the content of the data header, the reason that the core A stops running. Because the content of the data header includes the first default information, the debugging unit can determine, according to the first default information, that the reason that the core A stops running is a reason other than the preset event.

[0126] In an embodiment of the present invention, as shown in FIG. 8, the debugging unit includes a debugging agent and a debugger. It should be noted that, the debugging agent and the debugger in the debugging apparatus shown in FIG. 8 may be located in a same host machine, or may be located in different host machines. Referring to the remote debugging system shown in FIG. 2, the debugging agent and the debugger are respectively located in different host machines. The debugging agent is located in a first host machine, and the debugger is located in the second host machine.

[0127] If the debugging unit includes the debugging agent and the debugger, and the reason that the core A stops running is the preset event, the debugging agent is specifically configured to: after determining that the core A stops running, determine the reason that the core A stops running, and if the reason that the core A stops running is the preset event, report a second message to the debugger, where the second message includes an identifier indicating that the core A stops running and the identifier of the preset event.

[0128] Further, the debugging agent is specifically configured to: after determining, in a polling manner, that the core A stops running, determine, by reading the identifier

that is of the preset event and that is included in the content of the data header, that the reason that the core A stops running is the preset event; and report, to the debugger, the second message including the identifier indicating that the core A stops running and the identifier of the preset event.

[0129] The debugger is specifically configured to: after receiving the second message, obtain the debugging information of the preset event according to the identifier of the preset event, and set the preset pending breakpoint as the operation object of the preset event according to the debugging information of the preset event.

[0130] Further, the debugger is specifically configured to read, from the data body, the identifier of the operation object of the preset event or the loading address of the operation object of the preset event according to the identifier that is of the preset event and that is in the second message.

[0131] In an implementation manner of the present invention, if it is determined, according to the identifier of the preset event, that the preset event is the kernel module loading function, the process creation function, or the thread creation function, the debugger is specifically configured to: read, from the data body, the loading address of the operation object of the preset event according to the address of the data body located in the data header, and then set the preset pending breakpoint as the destination address of the operation object of the preset event according to the loading address of the operation object of the preset event.

[0132] In another implementation manner of the present invention, if it is determined, according to the identifier of the preset event, that the preset event is the process switching function, the process blocking function, the process wakeup function, the thread creation function, the thread switching function, or the thread blocking function, the debugger is specifically configured to: read, from the data body, the identifier of the operation object of the preset event according to the address of the data body located in the data header, determine the loading address of the operation object of the preset event according to the identifier of the operation object of the preset event, and set the preset pending breakpoint as the destination address of the operation object of the preset event according to the loading address of the operation object of the preset event.

[0133] It should be noted that, because the correspondence between the identifier of the operation object of the preset event and the loading address of the operation object of the preset event is pre-stored in the storage space of the second host machine, the debugger determines the loading address of the operation object of the preset event according to the identifier of the operation object of the preset event. Specifically, the debugger obtains, from the storage space of the second host machine, the loading address of the operation object of the preset event according to the identifier of the operation object of the preset event and the correspond-

ence.

[0134] In another embodiment of the present invention, if the debugging unit includes the debugging agent and the debugger, and the reason that the core A stops running is a reason other than the preset event, the debugging agent is specifically configured to: after determining that the core A stops running, determine the reason that the core A stops running, and if the reason that the core A stops running is a reason other than the preset event, send a first stop termination instruction to the core A, where the first stop termination instruction is used to instruct the core A to resume running, to collect the debugging information of the preset event.

[0135] Further, the debugging agent is specifically configured to: after determining, in a polling manner, that the core A stops running, read the first default information included in the content of the data header, and determine, according to the first default information, that the reason that the core A stops running is a reason other than the preset event.

[0136] In still another embodiment of the present invention, if the debugging unit includes the debugging agent and the debugger, and the reason that the core A stops running is a reason other than the preset event, the debugging agent is specifically configured to: after determining, in a polling manner, that the core A stops running, determine the reason that the core A stops running, and if the reason that the core A stops running is a reason other than the preset event, report a first message to the debugger, where the first message includes the identifier indicating that the core A stops running and a default identifier.

[0137] Further, the debugging agent is specifically configured to: after determining, in a polling manner, that the core A stops running, read the first default information included in the content of the data header; determine, according to the first default information, that the reason that the core A stops running is a reason other than the preset event; and report, to the debugger, the identifier indicating that the core A stops running and the default identifier.

[0138] The debugger is specifically configured to: after receiving the first message, instruct the debugging agent to send the first stop termination instruction to the core A, where the first stop termination instruction is used to instruct the core A to resume running.

[0139] The debugging agent is specifically configured to send the first stop termination instruction to the core A.

[0140] The debugging unit is further configured to instruct the core A to resume running, so that the core A knocks the pending breakpoint in a process of running the operation object of the preset event, and the core A enters a debugging state.

[0141] If the debugging unit includes the debugging agent and the debugger, the debugger is specifically configured to instruct the debugging agent to send a second stop termination instruction to the core A, where the second stop termination instruction is used to instruct the

core A to resume running, so that the core A knocks the pending breakpoint in process of running the operation object of the preset event, and the core A enters a debugging state.

[0142] It can be learned from above that, according to the debugging apparatus provided in this embodiment of the present invention, after determining that a core A of a multi-core processor on a target machine side stops running, if it is determined that a reason that the core A stops running is a preset event, the debugging apparatus obtains, from the target machine side, debugging information of the preset event, and sets a preset pending breakpoint as an operation object of the preset event according to the debugging information of the preset event, so as to debug the operation object of the preset event. In this embodiment of the present invention, because the preset event is a kernel mode code processing function or a user mode code processing function, the operation object of the preset event is kernel mode code or user mode code. Therefore, according to the debugging apparatus provided in this embodiment of the present invention, not only the kernel mode code can be debugged, but also the user mode code can be debugged.

Embodiment 4

[0143] Referring to FIG. 9, FIG. 9 is a structural block diagram of a debugging device 900 according to an embodiment of the present invention. The debugging device 900 is applied to the remote debugging system shown in FIG. 1, and the debugging device 900 is a specific implementation manner of a host machine in the remote debugging system shown in FIG. 1. Specifically, as shown in FIG. 9, the debugging device 900 includes: at least one processor 901, at least one network interface 904 or another user interface 903, a memory 905, and at least one communications bus 902. The communications bus 902 is configured to implement connection and communication between these components. The debugging device 900 optionally includes the user interface 903, including a display (such as a touchscreen, an LCD, a CTR, a holographic (Holographic) device, or a projector (Projector)), a keyboard, or a click device (such as a mouse, a trackball (trackball), a touchpad, or a touchscreen).

[0144] The memory 905 may include a read-only memory and a random access memory, and may provide an instruction and data for the processor. A part of the memory 905 may further include a nonvolatile random access memory (NVRAM).

[0145] In some embodiments, the memory 905 stores the following elements: an executable module or a data structure, or a subset thereof, or an extension set thereof:

an operating system 9051, including various system programs, such as a framework layer, a kernel library layer, and a drive layer, and configured to implement

various basic services and process hardware-based tasks; and

an application program module 9052, including various application programs, such as a launcher (launcher), a media player (Media Player), and a browser (Browser), and configured to implement various application services.

[0146] In this embodiment of the present invention, the processor 901 is configured to: after determining that the core A stops running, determine a reason that the core A stops running. A target machine located in the remote debugging system includes a multi-core processor, and the multi-core processor includes the core A.

[0147] It should be noted that, there is preset storage space in a memory of the target machine side, the preset storage space is specified by a debugging engineer in the memory of the target machine in advance, and both a size and an address of the preset storage space are fixed. The preset storage space is used to store a data header. Initially, the data header includes first default information and an address of a data body, and certainly, the first default information may be empty. The data body includes second default information, and certainly, the second default information may also be empty.

[0148] As shown in FIG. 6.2, if the core A stops running after completing execution of the debugging information collection function, in a process of executing the debugging information collection function, the core A writes, into the data header to replace default information, an identifier that is of a preset event and that is used for triggering the core A to execute the debugging information collection function, and writes an identifier of an operation object of the preset event into the data body. If the preset event is a kernel module loading function, a process creation function, or a thread creation function, the core A further writes a loading address of the operation object of the preset event into the data body. It should be noted that, the loading address of the operation object of the preset event may be continuous or discontinuous. As shown in FIG. 6.1, if the core A stops running due to another reason, the data header still includes the first default information and the address of the data body, and the data body still includes the second default information.

[0149] The processor 901 is specifically configured to: after determining that the core A stops running, determine, by reading content of the data header, the reason that the core A stops running; if the data header includes the identifier of the preset event, determine that the reason that the core A stops running is the preset event; and if the data header includes the first default information, determine that the reason that the core A stops running is a reason other than the preset event.

[0150] If the reason that the core A stops running is the preset event, the processor 901 is further configured to: obtain debugging information of the preset event, and set a preset pending breakpoint as the operation object

of the preset event according to the debugging information of the preset event, where the preset event is a kernel mode code processing function or a user mode code processing function, and the operation object of the preset event is kernel mode code or user mode code.

[0151] According to the foregoing description in this embodiment, it can be determined that the debugging information of the preset event includes the identifier of the preset event and the identifier of the operation object of the preset event. If the preset event is the kernel module loading function, the process creation function, or the thread creation function, the debugging information of the preset event further includes the loading address of the operation object of the preset event.

[0152] It should be noted that, the kernel mode code processing function is the kernel module loading function; and the user mode code processing function is the process creation function, a process switching function, a process blocking function, a process wakeup function, the thread creation function, a thread switching function, a thread blocking function, or a thread wakeup function. If the preset event is the kernel module loading function, the operation object of the preset event is a kernel module; if the preset event is the process creation function, the process switching function, the process blocking function, or the process wakeup function, the operation object of the preset event is a process; and if the preset event is the thread creation function, the thread switching function, the thread blocking function, or the thread wakeup function, the operation object of the preset event is a thread.

[0153] It should be noted that, if the reason that the core A stops running is the preset event, and if it is determined, according to the identifier of the preset event, that the preset event is the kernel module loading function, the process creation function, or the thread creation function, the processor 901 is specifically configured to: read, from the data body, the loading address of the operation object of the preset event according to the address of the data body located in the data header, and then set the preset pending breakpoint as a destination address of the operation object of the preset event according to the loading address of the operation object of the preset event.

[0154] Alternatively, if it is determined, according to the identifier of the preset event, that the preset event is the process switching function, the process blocking function, the process wakeup function, the thread creation function, the thread switching function, or the thread blocking function, the processor 901 is configured to: read, from the data body, the identifier of the operation object of the preset event according to the address of the data body located in the data header, determine the loading address of the operation object of the preset event according to the identifier of the operation object of the preset event, and set the preset pending breakpoint as a destination address of the operation object of the preset event according to the loading address of the operation

object of the preset event.

[0155] It should be noted that, a correspondence between the identifier of the operation object of the preset event and the loading address of the operation object of the preset event is pre-stored in storage space of the debugging device. Therefore, the processor 901 is specifically configured to obtain, from the storage space of the second host machine, the loading address of the operation object of the preset event according to the identifier of the operation object of the preset event and the correspondence.

[0156] The memory 905 is configured to store the debugging information of the preset event.

[0157] The processor 901 is further configured to instruct the core A to resume running, so that the core A knocks the pending breakpoint in a process of running the operation object of the preset event, and the core A enters a debugging state.

[0158] In another embodiment of the present invention, if the reason that the core A stops running is a reason other than the preset event, the processor 901 is further configured to instruct the core A to resume running, to collect the debugging information of the preset event.

[0159] It can be learned from above that, according to the debugging device provided in this embodiment of the present invention, after determining that a core A of a multi-core processor on a target machine side stops running, if it is determined that a reason that the core A stops running is a preset event, the debugging device obtains, from the target machine side, debugging information of the preset event, and sets a preset pending breakpoint as an operation object of the preset event according to the debugging information of the preset event, so as to debug the operation object of the preset event. In this embodiment of the present invention, because the preset event is a kernel mode code processing function or a user mode code processing function, the operation object of the preset event is kernel mode code or user mode code. Therefore, according to the debugging device provided in this embodiment of the present invention, not only the kernel mode code can be debugged, but also the user mode code can be debugged.

[0160] Mutual reference may be made to the foregoing embodiments. It may be clearly understood by a person skilled in the art that, for the purpose of convenient and brief description, for a detailed working process of the foregoing apparatus, device, and unit, reference may be made to a corresponding process in the foregoing method embodiments, and details are not described herein again.

[0161] In the several embodiments provided in this application, it should be understood that the disclosed method, apparatus, and device may be implemented in other manners. A person of ordinary skill in the art may be aware that, in combination with the examples described in the embodiments disclosed in this specification, units and algorithm steps may be implemented by electronic hardware or a combination of computer software and

electronic hardware. Whether the functions are performed by hardware or software depends on particular applications and design constraint conditions of the technical solutions. A person skilled in the art may use different methods to implement the described functions for each particular application, but it should not be considered that the implementation goes beyond the scope of the present invention.

[0162] It should be understood that, the technical solutions of the present invention essentially, or the part contributing to the prior art, or some of the technical solutions may be implemented in a form of a software product. The software product is stored in a storage medium, and includes several instructions for instructing a computer device (which may be a personal computer, a server, a network device, or the like) to perform all or some of the steps of the methods described in the embodiments of the present invention. The foregoing storage medium includes: any medium that can store program code, such as a USB flash drive, a removable hard disk, a read-only memory (ROM, Read-Only Memory), a random access memory (RAM, Random Access Memory), a magnetic disk, or an optical disc.

[0163] The foregoing descriptions are merely specific implementation manners of the present invention, but are not intended to limit the present invention.

Claims

1. A debugging method, applied to a target machine in a remote debugging system, wherein the target machine comprises a multi-core processor, and the method comprises:

stopping (S101) running, by a core A of the multi-core processor after completing execution of a processing routine of a preset event, and sending a running stop signal to other cores in a process of stopping running, wherein the running stop signal is used to instruct the other cores to stop running, the other cores refer to all cores that are located in the multi-core processor and that belong to a same logical group as the core A, and the preset event is a kernel mode code processing function or a user mode code processing function;

after receiving a first stop termination instruction and resuming running, executing (S103), by the core A, a debugging information collection function to collect debugging information of the preset event, and stopping running after completing the execution of the debugging information collection function, wherein the first stop termination instruction is sent by a debugging apparatus in the remote debugging system after the debugging apparatus determines that the core A stops running and that a reason that the core A

- stops running is a reason other than the preset event;
 after receiving a second stop termination instruction and resuming running, sending (S105), by the core A, a running resumption instruction to the other cores, wherein the running resumption instruction is used to instruct the other cores to resume running, and the second stop termination instruction is sent by the debugging apparatus after the debugging apparatus sets a preset pending breakpoint as an operation object of the preset event according to the debugging information of the preset event; and knocking (S105), by the core A, the pending breakpoint in a process of running the operation object of the preset event, so as to enter a debugging state.
2. The method according to claim 1, wherein the debugging information collection function comprises a debugging interface function, a first debugging exception instruction is set in a header of the debugging interface function, and the first debugging exception instruction is used to instruct the core A to stop running; and the step of stopping (S101) running, by the core A after completing execution of a processing routine of a preset event, and sending a running stop signal to other cores in a process of stopping running specifically comprises:
 after completing the execution of the processing routine of the preset event, starting, by the core A, to execute the first debugging exception instruction located in the header of the debugging interface function, and sending the running stop signal to the other cores in a process of executing the first debugging exception instruction.
3. The method according to claim 2, wherein the debugging information collection function further comprises a function associated with the preset event, a second debugging exception instruction is further set in a trailer of the debugging interface function, and the second debugging exception instruction is used to instruct the core A to stop running; and the step of executing (S103), by the core A, a debugging information collection function to collect debugging information of the preset event, and stopping running after completing the execution of the debugging information collection function specifically comprises:
 executing, by the core A, the function associated with the preset event to collect the debugging information of the preset event, and then executing the second debugging exception instruction located in the trailer of the debugging interface function.
4. The method according to claim 3, wherein
- the function associated with the preset event is located in a kernel debugging stub, and the kernel debugging stub is implanted into an operating system of the target machine in advance.
5. The method according to any one of claims 1 to 4, wherein
 the kernel mode code processing function is a kernel module loading function; and
 the user mode code processing function is a process creation function, a process switching function, a process blocking function, a process wakeup function, a thread creation function, a thread switching function, a thread blocking function, or a thread wakeup function.
6. The method according to claim 5, wherein
 if the preset event is the kernel module loading function, the operation object of the preset event is a kernel module;
 if the preset event is the process creation function, the process switching function, the process blocking function, or the process wakeup function, the operation object of the preset event is a process; and
 if the preset event is the thread creation function, the thread switching function, the thread blocking function, or the thread wakeup function, the operation object of the preset event is a thread.
7. The method according to claim 5 or 6, wherein the debugging information of the preset event comprises an identifier of the operation object of the preset event; and
 if the preset event is the kernel module loading function, the process creation function, or the thread creation function, the debugging information of the preset event further comprises a loading address of the operation object of the preset event.
8. The method according to any one of claims 1 to 7, wherein the multi-core processor is integrated into a system on chip SOC;
 the step of sending, by a core A, a running stop signal to other cores, wherein the running stop signal is used to instruct the other cores to stop running specifically comprises:
 sending, by the core A, the running stop signal to the other cores by using a hardware cross-trigger network of the SOC, wherein the running stop signal is used to instruct the other cores to stop running; and
 the step of sending (S105), by the core A, a running resumption instruction to the other cores, wherein the running resumption instruction is used to instruct the other cores to resume running specifically comprises:
 sending, by the core A, the running resumption

instruction to the other cores by using the hardware cross-trigger network of the SOC, wherein the running resumption instruction is used to instruct the other cores to resume running.

9. A debugging method, executed by a debugging apparatus deployed in a host machine, wherein the host machine is located in a remote debugging system, and the method comprises:

determining (S102), by the debugging apparatus after determining that a core A stops running, a reason that the core A stops running, wherein the core A belongs to a multi-core processor of a target machine in the remote debugging system;

if the reason that the core A stops running is a preset event, obtaining (S104), by the debugging apparatus, debugging information of the preset event, and setting a preset pending breakpoint as an operation object of the preset event according to the debugging information of the preset event, wherein the preset event is a kernel mode code processing function or a user mode code processing function, and the operation object of the preset event is kernel mode code or user mode code; and

instructing, by the debugging apparatus, the core A to resume running, so that the core A knocks the pending breakpoint in a process of running the operation object of the preset event, and the core A enters a debugging state.

10. The method according to claim 9, wherein if the reason that the core A stops running is a reason other than the preset event, instructing, by the debugging apparatus, the core A to resume running, to collect the debugging information of the preset event.

11. The method according to claim 9 or 10, wherein the function of the kernel mode code is a kernel module loading function; and the function of the user mode code is a process creation function, a process switching function, a process blocking function, a process wakeup function, a thread creation function, a thread switching function, a thread blocking function, or a thread wakeup function.

12. The method according to claim 11, wherein if the preset event is the kernel module loading function, the process creation function, or the thread creation function, the debugging information of the preset event comprises a loading address of the operation object of the preset event; and the step of obtaining, by the debugging apparatus, debugging information of the preset event, and set-

ting a preset pending breakpoint as an operation object of the preset event according to the debugging information of the preset event specifically comprises:

obtaining, by the debugging apparatus, the loading address of the operation object of the preset event, and setting the preset pending breakpoint as a destination address of the operation object of the preset event according to the loading address of the operation object of the preset event.

13. The method according to claim 11, wherein if the preset event is the process switching function, the process blocking function, the process wakeup function, the thread switching function, the thread blocking function, or the thread wakeup function, the debugging information of the preset event comprises an identifier of the operation object of the preset event; and

the step of obtaining (S104), by the debugging apparatus, debugging information of the preset event, and setting a preset pending breakpoint as an operation object of the preset event according to the debugging information of the preset event specifically comprises:

obtaining, by the debugging apparatus, the identifier of the operation object of the preset event, determining a loading address of the operation object of the preset event according to the identifier of the operation object of the preset event, and setting the preset pending breakpoint as a destination address of the operation object of the preset event according to the loading address of the operation object of the preset event.

14. The method according to claim 13, wherein a correspondence between the identifier of the operation object of the preset event and the loading address of the operation object of the preset event is pre-stored in storage space of the host machine; and the step of determining a loading address of the operation object of the preset event according to the identifier of the operation object of the preset event specifically comprises:

obtaining, from the storage space of the host machine, the loading address of the operation object of the preset event according to the identifier of the operation object of the preset event and the correspondence.

15. The method according to any one of claims 9 to 14, wherein the debugging apparatus comprises a debugger and a debugging agent; the step of determining, by the debugging apparatus after determining that a core A stops running, a reason that the core A stops running, and if the reason that the core A stops running is a preset event, obtaining, by the debugging apparatus, debugging in-

formation of the preset event, and setting a preset pending breakpoint as an operation object of the preset event according to the debugging information of the preset event specifically comprises:

after determining, in a polling manner, that the core A stops running, determining (S202), by the debugging agent, the reason that the core A stops running, and if the reason that the core A stops running is the preset event, reporting (S202), by the debugging agent, a second message to the debugger, wherein the second message comprises an identifier indicating that the core A stops running and an identifier of the preset event; and

after receiving the second message, obtaining, by the debugger, the debugging information of the preset event according to the identifier of the preset event, and setting the preset pending breakpoint as the operation object of the preset event according to the debugging information of the preset event; and

the step of instructing, by the debugging apparatus, the core A to resume running specifically comprises:

instructing, by the debugger, the debugging agent to send a second stop termination instruction to the core A, wherein the second stop termination instruction is used to instruct the core A to resume running.

16. The method according to claim 15, wherein the step of determining, by the debugging apparatus after determining that a core A stops running, a reason that the core A stops running, and if the reason that the core A stops running is a reason other than the preset event, instructing, by the debugging apparatus, the core A to resume running, to collect the debugging information of the preset event specifically comprises:

after determining, in a polling manner, that the core A stops running, determining (S202), by the debugging agent, the reason that the core A stops running, and if the reason that the core A stops running is a reason other than the preset event, sending, by the debugging agent, a first stop termination instruction to the core A, wherein the first stop termination instruction is used to instruct the core A to resume running, to collect the debugging information of the preset event.

17. The method according to any one of claims 9 to 15, wherein the debugging apparatus comprises the debugger and the debugging agent; and the step of determining, by the debugging apparatus after determining that a core A stops running, a reason that the core A stops running, and if the reason that the core A stops running is a reason other than

the preset event, instructing, by the debugging apparatus, the core A to resume running, to collect the debugging information of the preset event specifically comprises:

after determining, in a polling manner, that the core A stops running, determining (S202), by the debugging agent, the reason that the core A stops running, and if the reason that the core A stops running is a reason other than the preset event, reporting (S202) a first message to the debugger, wherein the first message comprises the identifier indicating that the core A stops running and a default identifier; and

after receiving the first message, instructing, by the debugger, the debugging agent to send a first stop termination instruction to the core A, wherein the first stop termination instruction is used to instruct the core A to resume running, to collect the debugging information of the preset event.

18. A multi-core processor, applied to a target machine in a remote debugging system, wherein the multi-core processor comprises a core A, other cores, and an input port, and the other cores refer to all cores that are located in the multi-core processor and that belong to a same logical group as the core A, wherein the core A is configured to: stop running after completing execution of a processing routine of a preset event, and send a running stop signal to the other cores in a process of stopping running, wherein the running stop signal is used to instruct the other cores to stop running, and the preset event is a kernel mode code processing function or a user mode code processing function;

the input port is configured to: receive a first stop termination instruction, and forward the first stop termination instruction to the core A, wherein the first stop termination instruction is sent by a debugging apparatus in the remote debugging system after the debugging apparatus determines that the core A stops running and that a reason that the core A stops running is a reason other than the preset event;

the core A is further configured to: after receiving the first stop termination instruction and resuming running, execute a debugging information collection function to collect debugging information of the preset event, and stop running after completing the execution of the debugging information collection function;

the input port is further configured to: receive a second stop termination instruction, and forward the second stop termination instruction to the core A, wherein the second stop termination instruction is sent by the debugging apparatus after the debugging apparatus sets a preset pending breakpoint as an operation object of the preset event according to the

- debugging information of the preset event; and the core A is further configured to: send a running resumption instruction to the other cores after receiving the second stop termination instruction and resuming running, wherein the running resumption instruction is used to instruct the other cores to resume running; and knock the pending breakpoint in a process of running the operation object of the preset event, so as to enter a debugging state.
19. The multi-core processor according to claim 18, wherein
the debugging information collection function comprises a debugging interface function, a first debugging exception instruction is set in a header of the debugging interface function, and the first debugging exception instruction is used to instruct the core A to stop running; and
the core A is specifically configured to: after completing the execution of the processing routine of the preset event, start to execute the first debugging exception instruction located in the header of the debugging interface function, and send the running stop signal to the other cores in a process of executing the first debugging exception instruction.
20. The multi-core processor according to claim 19, wherein
the debugging information collection function further comprises a function associated with the preset event, a second debugging exception instruction is further set in a trailer of the debugging interface function, and the second debugging exception instruction is used to instruct the core A to stop running; and
the core A is specifically configured to: execute the function associated with the preset event to collect the debugging information of the preset event, and then execute the second debugging exception instruction located in the trailer of the debugging interface function.
21. The multi-core processor according to claim 20, wherein
the function associated with the preset event is located in a kernel debugging stub, and the kernel debugging stub is implanted into an operating system of the target machine in advance.
22. The multi-core processor according to any one of claims 18 to 21, wherein the multi-core processor is integrated into a system on chip SOC;
the core A is specifically configured to send the running stop signal to the other cores by using a hardware cross-trigger network of the SOC, wherein the running stop signal is used to instruct the other cores to stop running; and
the core A is specifically configured to send the running resumption instruction to the other cores by using the hardware cross-trigger network of the SOC, wherein the running resumption instruction is used to instruct the other cores to resume running.
23. A debugging apparatus, wherein the debugging apparatus is deployed on a host machine in a remote debugging system, and the debugging apparatus comprises a debugging unit and a receiving unit, wherein
the receiving unit is configured to: receive a message used for instructing a core A to stop running, and forward the message used for instructing the core A to stop running to the debugging unit, wherein a target machine located in the remote debugging system comprises a multi-core processor, and the multi-core processor comprises the core A;
the debugging unit is configured to: after determining that the core A stops running, determine a reason that the core A stops running;
if the reason that the core A stops running is a preset event, the debugging unit is further configured to: obtain debugging information of the preset event, and set a preset pending breakpoint as an operation object of the preset event according to the debugging information of the preset event, wherein the preset event is a kernel mode code processing function or a user mode code processing function, and the operation object of the preset event is kernel mode code or user mode code; and
the debugging unit is further configured to instruct the core A to resume running, so that the core A knocks the pending breakpoint in a process of running the operation object of the preset event, and the core A enters a debugging state.
24. The apparatus according to claim 23, wherein
if the reason that the core A stops running is a reason other than the preset event, the debugging unit is further configured to instruct the core A to resume running, to collect the debugging information of the preset event.
25. The apparatus according to claim 23 or 24, wherein
the function of the kernel mode code is a kernel module loading function; and
the function of the user mode code is a process creation function, a process switching function, a process blocking function, a process wakeup function, a thread creation function, a thread switching function, a thread blocking function, or a thread wakeup function.
26. The apparatus according to claim 25, wherein
if the preset event is the kernel module loading function, the process creation function, or the thread creation function, the debugging information of the preset event comprises a loading address of the operation object of the preset event; and

the debugging unit is specifically configured to: obtain the loading address of the operation object of the preset event, and set the preset pending breakpoint as a destination address of the operation object of the preset event according to the loading address of the operation object of the preset event.

27. The apparatus according to claim 25, wherein if the preset event is the process switching function, the process blocking function, the process wakeup function, the thread switching function, the thread blocking function, or the thread wakeup function, the debugging information of the preset event comprises an identifier of the operation object of the preset event; and the debugging unit is specifically configured to: obtain the identifier of the operation object of the preset event, determine a loading address of the operation object of the preset event according to the identifier of the operation object of the preset event, and set the preset pending breakpoint as a destination address of the operation object of the preset event according to the loading address of the operation object of the preset event.
28. The apparatus according to claim 27, wherein a correspondence between the identifier of the operation object of the preset event and the loading address of the operation object of the preset event is pre-stored in storage space of the host machine; and the debugging unit is specifically configured to obtain, from the storage space of the host machine, the loading address of the operation object of the preset event according to the identifier of the operation object of the preset event and the correspondence.
29. The apparatus according to any one of claims 23 to 28, wherein the debugging unit comprises a debugger and a debugging agent; the debugging agent is specifically configured to: after determining that the core A stops running, determine the reason that the core A stops running, and if the reason that the core A stops running is the preset event, report a second message to the debugger, wherein the second message comprises an identifier indicating that the core A stops running and an identifier of the preset event; the debugger is specifically configured to: after receiving the second message, obtain the debugging information of the preset event according to the identifier of the preset event, and set the preset pending breakpoint as the operation object of the preset event according to the debugging information of the preset event; and the debugger is specifically configured to instruct the debugging agent to send a second stop termination instruction to the core A, wherein the second stop termination instruction is used to instruct the core A

to resume running.

30. The apparatus according to claim 29, wherein the debugging agent is specifically configured to: after determining that the core A stops running, determine the reason that the core A stops running, and if the reason that the core A stops running is a reason other than the preset event, send a first stop termination instruction to the core A, wherein the first stop termination instruction is used to instruct the core A to resume running, to collect the debugging information of the preset event.
31. The apparatus according to any one of claims 23 to 29, wherein the debugging apparatus further comprises the debugger and the debugging agent; the debugging agent is specifically configured to: after determining that the core A stops running, determine the reason that the core A stops running, and if the reason that the core A stops running is a reason other than the preset event, report a first message to the debugger, wherein the first message comprises the identifier indicating that the core A stops running and a default identifier; and the debugger is specifically configured to: after receiving the first message, instruct the debugging agent to send a first stop termination instruction to the core A, wherein the first stop termination instruction is used to instruct the core A to resume running, to collect the debugging information of the preset event.

Patentansprüche

1. "Debugging"-Verfahren, das bei einer Zielmaschine in einem Fern-"Debugging"-System angewendet wird, wobei die Zielmaschine einen Mehrkernprozessor umfasst, und das Verfahren Folgendes umfasst:

Stoppen (S101) der Ausführung, durch einen Kern A des Mehrkernprozessors nach der Beendigung der Durchführung einer Verarbeitungsroutine eines voreingestellten Ereignisses, und Senden eines Ausführungsstoppsignals zu anderen Kernen in einem Prozess des Stoppens der Ausführung, wobei das Ausführungsstoppsignal dafür verwendet wird, die anderen Kerne anzuweisen, die Ausführung zu stoppen, wobei sich die anderen Kerne auf alle Kerne beziehen, die sich im Mehrkernprozessor befinden und zu derselben logischen Gruppe wie der Kern A gehören, und das voreingestellte Ereignis eine Kernel-Moduscode-Verarbeitungsfunktion oder eine Benutzermoduscode-Verarbeitungsfunktion ist; nach dem Empfang einer ersten Stoppab-

- bruchsanweisung und der Wiederaufnahme der Ausführung, Durchführen (S103), durch den Kern A, einer "Debugging"-Informationen-Sammelfunktion zum Sammeln von "Debugging"-Informationen des voreingestellten Ereignisses, und Stoppen der Ausführung nach Beendigung der Durchführung der "Debugging"-Informationen-Sammelfunktion, wobei die erste Stoppabbruchsanweisung durch eine "Debugging"-Vorrichtung im Fern-"Debugging"-System gesendet wird, nachdem die "Debugging"-Vorrichtung bestimmt, dass der Kern A die Ausführung stoppt und dass ein Grund dafür, dass der Kern A die Ausführung stoppt, ein Grund außer dem voreingestellten Ereignis ist; nach dem Empfang einer zweiten Stoppabbruchsanweisung und der Wiederaufnahme der Ausführung, Senden (S105), durch den Kern A, einer Ausführungswiederaufnahmeanweisung zu den anderen Kernen, wobei die Ausführungswiederaufnahmeanweisung dafür verwendet wird, die anderen Kerne anzuweisen, die Ausführung wiederaufzunehmen, und die zweite Stoppabbruchsanweisung durch die "Debugging"-Vorrichtung gesendet wird, nachdem die "Debugging"-Vorrichtung einen voreingestellten bevorstehenden Haltepunkt als ein Operationsobjekt des voreingestellten Ereignisses gemäß den "Debugging"-Informationen des voreingestellten Ereignisses einstellt; und "Knocking" (S105), durch den Kern A, des bevorstehenden Haltepunktes in einem Prozess der Ausführung des Operationsobjekts des voreingestellten Ereignisses, sodass in einen "Debugging"-Zustand eingetreten wird.
2. Verfahren nach Anspruch 1, wobei die "Debugging"-Informationen-Sammelfunktion eine "Debugging"-Schnittstellenfunktion umfasst, eine erste "Debugging"-Ausnahmeanweisung in einem "Header" der "Debugging"-Schnittstellenfunktion eingestellt ist und die erste "Debugging"-Ausnahmeanweisung dafür verwendet wird, den Kern A anzuweisen, die Ausführung zu stoppen; und der Schritt des Stoppens (S101) der Ausführung, durch den Kern A nach der Beendigung der Durchführung einer Verarbeitungsroutine eines voreingestellten Ereignisses, und des Sendens eines Ausführungsstoppsignals zu anderen Kernen in einem Prozess des Stoppens der Ausführung spezifisch Folgendes umfasst:
nach der Beendigung der Durchführung der Verarbeitungsroutine des voreingestellten Ereignisses, Starten, durch den Kern A, der Durchführung der ersten "Debugging"-Ausnahmeanweisung, die sich im "Header" der "Debugging"-Schnittstellenfunktion befindet, und Senden des Ausführungsstoppsignals zu den anderen Kernen in einem Prozess des Durch-
- führens der ersten "Debugging"-Ausnahmeanweisung.
3. Verfahren nach Anspruch 2, wobei die "Debugging"-Informationen-Sammelfunktion ferner eine mit dem voreingestellten Ereignis assoziierte Funktion umfasst, eine zweite "Debugging"-Ausnahmeanweisung ferner in einem "Trailer" der "Debugging"-Schnittstellenfunktion eingestellt ist und die zweite "Debugging"-Ausnahmeanweisung dafür verwendet wird, den Kern A anzuweisen, die Ausführung zu stoppen; und der Schritt des Durchführens (S103), durch den Kern A, einer "Debugging"-Informationen-Sammelfunktion zum Sammeln von "Debugging"-Informationen des voreingestellten Ereignisses, und des Stoppens der Ausführung nach Beendigung der Durchführung der "Debugging"-Informationen-Sammelfunktion spezifisch Folgendes umfasst:
Durchführen, durch den Kern A, der mit dem voreingestellten Ereignis assoziierten Funktion, um die "Debugging"-Informationen des voreingestellten Ereignisses zu sammeln, und dann Durchführen der zweiten "Debugging"-Ausnahmeanweisung, die sich im "Trailer" der "Debugging"-Schnittstellenfunktion befindet.
4. Verfahren nach Anspruch 3, wobei sich die mit dem voreingestellten Ereignis assoziierte Funktion in einem Kernel-"Debugging-Stub" befindet und das Kernel-"Debugging-Stub" im Voraus in ein Betriebssystem der Zielmaschine implantiert wird.
5. Verfahren nach einem der Ansprüche 1 bis 4, wobei die Kernel-Moduscode-Verarbeitungsfunktion eine Kernel-Modulladefunktion ist; und die Benutzermoduscode-Verarbeitungsfunktion eine Prozesserzeugungsfunktion, eine Prozessschaltfunktion, eine Prozessblockierfunktion, eine Prozessaufweckfunktion, eine Thread-Erzeugungsfunktion, eine Thread-Schaltfunktion, eine Thread-Blockierfunktion oder eine Thread-Aufweckfunktion ist.
6. Verfahren nach Anspruch 5, wobei falls das voreingestellte Ereignis die Kernel-Modulladefunktion ist, das Operationsobjekt des voreingestellten Ereignisses ein Kernel-Modul ist; falls das voreingestellte Ereignis die Prozesserzeugungsfunktion, die Prozessschaltfunktion, die Prozessblockierfunktion oder die Prozessaufweckfunktion ist, das Operationsobjekt des voreingestellten Ereignisses ein Prozess ist; und falls das voreingestellte Ereignis die Thread-Erzeugungsfunktion, die Thread-Schaltfunktion, die Thread-Blockierfunktion oder die Thread-Aufweckfunktion ist, das Operationsobjekt des voreingestell-

ten Ereignisses ein Thread ist.

7. Verfahren nach Anspruch 5 oder 6, wobei die "Debugging"-Informationen des voreingestellten Ereignisses eine Kennung des Operationsobjekts des voreingestellten Ereignisses umfassen; und falls das voreingestellte Ereignis die Kernel-Modul-ladefunktion, die Prozesserzeugungsfunktion oder die Thread-Erzeugungsfunktion ist, die "Debugging"-Informationen des voreingestellten Ereignisses ferner eine Ladeadresse des Operationsobjekts des voreingestellten Ereignisses umfassen. 5 10
8. Verfahren nach einem der Ansprüche 1 bis 7, wobei der Mehrkernprozessor in ein System-auf-Chip bzw. SOC ("System On Chip") integriert ist; der Schritt des Sendens, durch einen Kern A, eines Ausführungsstoppsignals zu anderen Kernen, wobei das Ausführungsstoppsignal dafür verwendet wird, die anderen Kerne anzuweisen, die Ausführung zu stoppen, spezifisch Folgendes umfasst: 15 20

Senden, durch den Kern A, des Ausführungsstoppsignals zu den anderen Kernen unter Verwendung eines Hardware-"Cross-Trigger"-Netzwerks des SOC, wobei das Ausführungsstoppsignal dafür verwendet wird, die anderen Kerne anzuweisen, die Ausführung zu stoppen; und der Schritt des Sendens (S105), durch den Kern A, einer Ausführungswiederaufnahmeanweisung zu den anderen Kernen, wobei die Ausführungswiederaufnahmeanweisung dafür verwendet wird, die anderen Kerne anzuweisen, die Ausführung wiederaufzunehmen, spezifisch Folgendes umfasst: 25 30

Senden, durch den Kern A, der Ausführungswiederaufnahmeanweisung zu den anderen Kernen unter Verwendung des Hardware-"Cross-Trigger"-Netzwerks des SOC, wobei die Ausführungswiederaufnahmeanweisung dafür verwendet wird, die anderen Kerne anzuweisen, die Ausführung wiederaufzunehmen. 35 40

9. "Debugging"-Verfahren, das durch eine "Debugging"-Vorrichtung ausgeführt wird, die in einer Host-Maschine eingesetzt wird, wobei sich die Host-Maschine in einem Fern-"Debugging"-System befindet, und das Verfahren Folgendes umfasst: 45 50

Bestimmen (S102), durch die "Debugging"-Vorrichtung nach der Bestimmung, dass ein Kern A die Ausführung stoppt, eines Grundes dafür, dass der Kern A die Ausführung stoppt, wobei der Kern A zu einem Mehrkernprozessor einer Zielmaschine im Fern-"Debugging"-System gehört; 55

falls der Grund dafür, dass der Kern A die Ausführung stoppt, ein voreingestelltes Ereignis ist, Erhalten (S104), durch die "Debugging"-Vorrichtung, von "Debugging"-Informationen des voreingestellten Ereignisses und Einstellen eines voreingestellten bevorstehenden Haltepunktes als ein Operationsobjekt des voreingestellten Ereignisses gemäß den "Debugging"-Informationen des voreingestellten Ereignisses, wobei das des voreingestellte Ereignis eine Kernel-Moduscode-Verarbeitungsfunktion oder eine Benutzermoduscode-Verarbeitungsfunktion ist und das Operationsobjekt des voreingestellten Ereignisses ein Kernel-Moduscode oder ein Benutzermoduscode ist; und Anweisen, durch die "Debugging"-Vorrichtung, des Kerns A, die Ausführung wiederaufzunehmen, sodass der Kern A den bevorstehenden Haltepunkt in einem Prozess der Ausführung des Operationsobjekts des voreingestellten Ereignisses "knockt" und der Kern A in einen "Debugging"-Zustand eintritt.

10. Verfahren nach Anspruch 9, wobei falls der Grund dafür, dass der Kern A die Ausführung stoppt, ein Grund außer dem voreingestellten Ereignis ist, Anweisen, durch die "Debugging"-Vorrichtung, des Kerns A, die Ausführung wiederaufzunehmen, um die "Debugging"-Informationen des voreingestellten Ereignisses zu sammeln. 25 30
 11. Verfahren nach Anspruch 9 oder 10, wobei die Funktion des Kernel-Moduscodes eine Kernel-Modulladefunktion ist; und die Funktion des Benutzermoduscodes eine Prozesserzeugungsfunktion, eine Prozessschaltfunktion, eine Prozessblockierfunktion, eine Prozessaufweckfunktion, eine Thread-Erzeugungsfunktion, eine Thread-Schaltfunktion, eine Thread-Blockierfunktion oder eine Thread-Aufweckfunktion ist. 35 40
 12. Verfahren nach Anspruch 11, wobei falls das voreingestellte Ereignis die Kernel-Modul-ladefunktion, die Prozesserzeugungsfunktion oder die Thread-Erzeugungsfunktion ist, die "Debugging"-Informationen des voreingestellten Ereignisses eine Ladeadresse des Operationsobjekts des voreingestellten Ereignisses umfassen; und der Schritt des Erhaltens, durch die "Debugging"-Vorrichtung, von "Debugging"-Informationen des voreingestellten Ereignisses und des Einstellens eines voreingestellten bevorstehenden Haltepunktes als ein Operationsobjekt des voreingestellten Ereignisses gemäß den "Debugging"-Informationen des voreingestellten Ereignisses spezifisch Folgendes umfasst: 45 50
- Erhalten, durch die "Debugging"-Vorrichtung, der Ladeadresse des Operationsobjekts des voreinge-

stellten Ereignisses und Einstellen des bevorstehenden Haltepunktes als eine Zieladresse des Operationsobjekts des voreingestellten Ereignisses gemäß der Ladeadresse des Operationsobjekts des voreingestellten Ereignisses.

13. Verfahren nach Anspruch 11, wobei falls das voreingestellte Ereignis die Prozessschaltfunktion, die Prozessblockierfunktion, die Prozessaufweckfunktion, die Thread-Schaltfunktion, die Thread-Blockierfunktion oder die Thread-Aufweckfunktion ist, die "Debugging"-Informationen des voreingestellten Ereignisses eine Kennung des Operationsobjekts des voreingestellten Ereignisses umfassen; und
 5
 10
 15
 20
 25
 30
 35
 40
 45
 50
 55
 60
 65
 70
 75
 80
 85
 90
 95
 100
 105
 110
 115
 120
 125
 130
 135
 140
 145
 150
 155
 160
 165
 170
 175
 180
 185
 190
 195
 200
 205
 210
 215
 220
 225
 230
 235
 240
 245
 250
 255
 260
 265
 270
 275
 280
 285
 290
 295
 300
 305
 310
 315
 320
 325
 330
 335
 340
 345
 350
 355
 360
 365
 370
 375
 380
 385
 390
 395
 400
 405
 410
 415
 420
 425
 430
 435
 440
 445
 450
 455
 460
 465
 470
 475
 480
 485
 490
 495
 500
 505
 510
 515
 520
 525
 530
 535
 540
 545
 550
 555
 560
 565
 570
 575
 580
 585
 590
 595
 600
 605
 610
 615
 620
 625
 630
 635
 640
 645
 650
 655
 660
 665
 670
 675
 680
 685
 690
 695
 700
 705
 710
 715
 720
 725
 730
 735
 740
 745
 750
 755
 760
 765
 770
 775
 780
 785
 790
 795
 800
 805
 810
 815
 820
 825
 830
 835
 840
 845
 850
 855
 860
 865
 870
 875
 880
 885
 890
 895
 900
 905
 910
 915
 920
 925
 930
 935
 940
 945
 950
 955
 960
 965
 970
 975
 980
 985
 990
 995
 1000
 1005
 1010
 1015
 1020
 1025
 1030
 1035
 1040
 1045
 1050
 1055
 1060
 1065
 1070
 1075
 1080
 1085
 1090
 1095
 1100
 1105
 1110
 1115
 1120
 1125
 1130
 1135
 1140
 1145
 1150
 1155
 1160
 1165
 1170
 1175
 1180
 1185
 1190
 1195
 1200
 1205
 1210
 1215
 1220
 1225
 1230
 1235
 1240
 1245
 1250
 1255
 1260
 1265
 1270
 1275
 1280
 1285
 1290
 1295
 1300
 1305
 1310
 1315
 1320
 1325
 1330
 1335
 1340
 1345
 1350
 1355
 1360
 1365
 1370
 1375
 1380
 1385
 1390
 1395
 1400
 1405
 1410
 1415
 1420
 1425
 1430
 1435
 1440
 1445
 1450
 1455
 1460
 1465
 1470
 1475
 1480
 1485
 1490
 1495
 1500
 1505
 1510
 1515
 1520
 1525
 1530
 1535
 1540
 1545
 1550
 1555
 1560
 1565
 1570
 1575
 1580
 1585
 1590
 1595
 1600
 1605
 1610
 1615
 1620
 1625
 1630
 1635
 1640
 1645
 1650
 1655
 1660
 1665
 1670
 1675
 1680
 1685
 1690
 1695
 1700
 1705
 1710
 1715
 1720
 1725
 1730
 1735
 1740
 1745
 1750
 1755
 1760
 1765
 1770
 1775
 1780
 1785
 1790
 1795
 1800
 1805
 1810
 1815
 1820
 1825
 1830
 1835
 1840
 1845
 1850
 1855
 1860
 1865
 1870
 1875
 1880
 1885
 1890
 1895
 1900
 1905
 1910
 1915
 1920
 1925
 1930
 1935
 1940
 1945
 1950
 1955
 1960
 1965
 1970
 1975
 1980
 1985
 1990
 1995
 2000
 2005
 2010
 2015
 2020
 2025
 2030
 2035
 2040
 2045
 2050
 2055
 2060
 2065
 2070
 2075
 2080
 2085
 2090
 2095
 2100
 2105
 2110
 2115
 2120
 2125
 2130
 2135
 2140
 2145
 2150
 2155
 2160
 2165
 2170
 2175
 2180
 2185
 2190
 2195
 2200
 2205
 2210
 2215
 2220
 2225
 2230
 2235
 2240
 2245
 2250
 2255
 2260
 2265
 2270
 2275
 2280
 2285
 2290
 2295
 2300
 2305
 2310
 2315
 2320
 2325
 2330
 2335
 2340
 2345
 2350
 2355
 2360
 2365
 2370
 2375
 2380
 2385
 2390
 2395
 2400
 2405
 2410
 2415
 2420
 2425
 2430
 2435
 2440
 2445
 2450
 2455
 2460
 2465
 2470
 2475
 2480
 2485
 2490
 2495
 2500
 2505
 2510
 2515
 2520
 2525
 2530
 2535
 2540
 2545
 2550
 2555
 2560
 2565
 2570
 2575
 2580
 2585
 2590
 2595
 2600
 2605
 2610
 2615
 2620
 2625
 2630
 2635
 2640
 2645
 2650
 2655
 2660
 2665
 2670
 2675
 2680
 2685
 2690
 2695
 2700
 2705
 2710
 2715
 2720
 2725
 2730
 2735
 2740
 2745
 2750
 2755
 2760
 2765
 2770
 2775
 2780
 2785
 2790
 2795
 2800
 2805
 2810
 2815
 2820
 2825
 2830
 2835
 2840
 2845
 2850
 2855
 2860
 2865
 2870
 2875
 2880
 2885
 2890
 2895
 2900
 2905
 2910
 2915
 2920
 2925
 2930
 2935
 2940
 2945
 2950
 2955
 2960
 2965
 2970
 2975
 2980
 2985
 2990
 2995
 3000
 3005
 3010
 3015
 3020
 3025
 3030
 3035
 3040
 3045
 3050
 3055
 3060
 3065
 3070
 3075
 3080
 3085
 3090
 3095
 3100
 3105
 3110
 3115
 3120
 3125
 3130
 3135
 3140
 3145
 3150
 3155
 3160
 3165
 3170
 3175
 3180
 3185
 3190
 3195
 3200
 3205
 3210
 3215
 3220
 3225
 3230
 3235
 3240
 3245
 3250
 3255
 3260
 3265
 3270
 3275
 3280
 3285
 3290
 3295
 3300
 3305
 3310
 3315
 3320
 3325
 3330
 3335
 3340
 3345
 3350
 3355
 3360
 3365
 3370
 3375
 3380
 3385
 3390
 3395
 3400
 3405
 3410
 3415
 3420
 3425
 3430
 3435
 3440
 3445
 3450
 3455
 3460
 3465
 3470
 3475
 3480
 3485
 3490
 3495
 3500
 3505
 3510
 3515
 3520
 3525
 3530
 3535
 3540
 3545
 3550
 3555
 3560
 3565
 3570
 3575
 3580
 3585
 3590
 3595
 3600
 3605
 3610
 3615
 3620
 3625
 3630
 3635
 3640
 3645
 3650
 3655
 3660
 3665
 3670
 3675
 3680
 3685
 3690
 3695
 3700
 3705
 3710
 3715
 3720
 3725
 3730
 3735
 3740
 3745
 3750
 3755
 3760
 3765
 3770
 3775
 3780
 3785
 3790
 3795
 3800
 3805
 3810
 3815
 3820
 3825
 3830
 3835
 3840
 3845
 3850
 3855
 3860
 3865
 3870
 3875
 3880
 3885
 3890
 3895
 3900
 3905
 3910
 3915
 3920
 3925
 3930
 3935
 3940
 3945
 3950
 3955
 3960
 3965
 3970
 3975
 3980
 3985
 3990
 3995
 4000
 4005
 4010
 4015
 4020
 4025
 4030
 4035
 4040
 4045
 4050
 4055
 4060
 4065
 4070
 4075
 4080
 4085
 4090
 4095
 4100
 4105
 4110
 4115
 4120
 4125
 4130
 4135
 4140
 4145
 4150
 4155
 4160
 4165
 4170
 4175
 4180
 4185
 4190
 4195
 4200
 4205
 4210
 4215
 4220
 4225
 4230
 4235
 4240
 4245
 4250
 4255
 4260
 4265
 4270
 4275
 4280
 4285
 4290
 4295
 4300
 4305
 4310
 4315
 4320
 4325
 4330
 4335
 4340
 4345
 4350
 4355
 4360
 4365
 4370
 4375
 4380
 4385
 4390
 4395
 4400
 4405
 4410
 4415
 4420
 4425
 4430
 4435
 4440
 4445
 4450
 4455
 4460
 4465
 4470
 4475
 4480
 4485
 4490
 4495
 4500
 4505
 4510
 4515
 4520
 4525
 4530
 4535
 4540
 4545
 4550
 4555
 4560
 4565
 4570
 4575
 4580
 4585
 4590
 4595
 4600
 4605
 4610
 4615
 4620
 4625
 4630
 4635
 4640
 4645
 4650
 4655
 4660
 4665
 4670
 4675
 4680
 4685
 4690
 4695
 4700
 4705
 4710
 4715
 4720
 4725
 4730
 4735
 4740
 4745
 4750
 4755
 4760
 4765
 4770
 4775
 4780
 4785
 4790
 4795
 4800
 4805
 4810
 4815
 4820
 4825
 4830
 4835
 4840
 4845
 4850
 4855
 4860
 4865
 4870
 4875
 4880
 4885
 4890
 4895
 4900
 4905
 4910
 4915
 4920
 4925
 4930
 4935
 4940
 4945
 4950
 4955
 4960
 4965
 4970
 4975
 4980
 4985
 4990
 4995
 5000
 5005
 5010
 5015
 5020
 5025
 5030
 5035
 5040
 5045
 5050
 5055
 5060
 5065
 5070
 5075
 5080
 5085
 5090
 5095
 5100
 5105
 5110
 5115
 5120
 5125
 5130
 5135
 5140
 5145
 5150
 5155
 5160
 5165
 5170
 5175
 5180
 5185
 5190
 5195
 5200
 5205
 5210
 5215
 5220
 5225
 5230
 5235
 5240
 5245
 5250
 5255
 5260
 5265
 5270
 5275
 5280
 5285
 5290
 5295
 5300
 5305
 5310
 5315
 5320
 5325
 5330
 5335
 5340
 5345
 5350
 5355
 5360
 5365
 5370
 5375
 5380
 5385
 5390
 5395
 5400
 5405
 5410
 5415
 5420
 5425
 5430
 5435
 5440
 5445
 5450
 5455
 5460
 5465
 5470
 5475
 5480
 5485
 5490
 5495
 5500
 5505
 5510
 5515
 5520
 5525
 5530
 5535
 5540
 5545
 5550
 5555
 5560
 5565
 5570
 5575
 5580
 5585
 5590
 5595
 5600
 5605
 5610
 5615
 5620
 5625
 5630
 5635
 5640
 5645
 5650
 5655
 5660
 5665
 5670
 5675
 5680
 5685
 5690
 5695
 5700
 5705
 5710
 5715
 5720
 5725
 5730
 5735
 5740
 5745
 5750
 5755
 5760
 5765
 5770
 5775
 5780
 5785
 5790
 5795
 5800
 5805
 5810
 5815
 5820
 5825
 5830
 5835
 5840
 5845
 5850
 5855
 5860
 5865
 5870
 5875
 5880
 5885
 5890
 5895
 5900
 5905
 5910
 5915
 5920
 5925
 5930
 5935
 5940
 5945
 5950
 5955
 5960
 5965
 5970
 5975
 5980
 5985
 5990
 5995
 6000
 6005
 6010
 6015
 6020
 6025
 6030
 6035
 6040
 6045
 6050
 6055
 6060
 6065
 6070
 6075
 6080
 6085
 6090
 6095
 6100
 6105
 6110
 6115
 6120
 6125
 6130
 6135
 6140
 6145
 6150
 6155
 6160
 6165
 6170
 6175
 6180
 6185
 6190
 6195
 6200
 6205
 6210
 6215
 6220
 6225
 6230
 6235
 6240
 6245
 6250
 6255
 6260
 6265
 6270
 6275
 6280
 6285
 6290
 6295
 6300
 6305
 6310
 6315
 6320
 6325
 6330
 6335
 6340
 6345
 6350
 6355
 6360
 6365
 6370
 6375
 6380
 6385
 6390
 6395
 6400
 6405
 6410
 6415
 6420
 6425
 6430
 6435
 6440
 6445
 6450
 6455
 6460
 6465
 6470
 6475
 6480
 6485
 6490
 6495
 6500
 6505
 6510
 6515
 6520
 6525
 6530
 6535
 6540
 6545
 6550
 6555
 6560
 6565
 6570
 6575
 6580
 6585
 6590
 6595
 6600
 6605
 6610
 6615
 6620
 6625
 6630
 6635
 6640
 6645
 6650
 6655
 6660
 6665
 6670
 6675
 6680
 6685
 6690
 6695
 6700
 6705
 6710
 6715
 6720
 6725
 6730
 6735
 6740
 6745
 6750
 6755
 6760
 6765
 6770
 6775
 6780
 6785
 6790
 6795
 6800
 6805
 6810
 6815
 6

und, falls der Grund dafür, dass der Kern A die Ausführung stoppt, ein Grund außer dem voreingestellten Ereignis ist, Senden, durch den "Debugging"-Agenten, einer ersten Stoppabbruchsanweisung zum Kern A, wobei die erste Stoppabbruchsanweisung dafür verwendet wird, den Kern A anzuweisen, die Ausführung wiederaufzunehmen, um die "Debugging"-Informationen des voreingestellten Ereignisses zu sammeln.

17. Verfahren nach einem der Ansprüche 9 bis 15, wobei die "Debugging"-Vorrichtung den Debugger und den "Debugging"-Agenten umfasst; und der Schritt des Bestimmens, durch die "Debugging"-Vorrichtung nach der Bestimmung, dass der Kern A die Ausführung stoppt, eines Grundes dafür, dass der Kern A die Ausführung stoppt, und, falls der Grund dafür, dass der Kern A die Ausführung stoppt, ein Grund außer dem voreingestellten Ereignis ist, des Anweisens, durch die "Debugging"-Vorrichtung, des Kerns A, die Ausführung wiederaufzunehmen, um die "Debugging"-Informationen des voreingestellten Ereignisses zu sammeln, spezifisch Folgendes umfasst:

nach der Bestimmung, auf eine "Polling"-Weise, dass der Kern A die Ausführung stoppt, Bestimmen (S202), durch den "Debugging"-Agenten, des Grundes dafür, dass der Kern A die Ausführung stoppt, und, falls der Grund dafür, dass der Kern A die Ausführung stoppt, ein Grund außer dem voreingestellten Ereignis ist, Melden (S202) einer ersten Nachricht zum Debugger, wobei die erste Nachricht die Kennung, die das Stoppen der Ausführung durch den Kern A angibt, und eine Standardkennung umfasst; und nach dem Empfang der ersten Nachricht, Anweisen, durch den Debugger, des "Debugging"-Agenten, eine erste Stoppabbruchsanweisung zum Kern A zu senden, wobei die erste Stoppabbruchsanweisung dafür verwendet wird, den Kern A anzuweisen, die Ausführung wiederaufzunehmen, um die "Debugging"-Informationen des voreingestellten Ereignisses zu sammeln.

18. Mehrkernprozessor, der an einer Zielmaschine in einem Fern-"Debugging"-System angewendet wird, wobei der Mehrkernprozessor einen Kern A, andere Kerne und einen Eingangsport umfasst und sich die anderen Kerne auf alle Kerne beziehen, die sich im Mehrkernprozessor befinden und zu derselben logischen Gruppe wie der Kern A gehören, wobei der Kern A ausgelegt ist zum: Stoppen der Ausführung nach der Beendigung der Durchführung einer Verarbeitungsroutine eines voreingestellten Ereignisses und Senden eines Ausführungsstoppsignals zu den anderen Kernen in einem Prozess des Stoppens der Ausführung, wobei das Ausführungsstopp-

signal dafür verwendet wird, die anderen Kerne anzuweisen, die Ausführung zu stoppen, und das voreingestellte Ereignis eine Kernel-Moduscode-Verarbeitungsfunktion oder eine Benutzermoduscode-Verarbeitungsfunktion ist;

der Eingangsport ausgelegt ist zum: Empfangen einer ersten Stoppabbruchsanweisung und Weiterleiten der ersten Stoppabbruchsanweisung zum Kern A, wobei die erste Stoppabbruchsanweisung durch eine "Debugging"-Vorrichtung im Fern-"Debugging"-System gesendet wird, nachdem die "Debugging"-Vorrichtung bestimmt, dass der Kern A die Ausführung stoppt, und dass ein Grund dafür, dass der Kern A die Ausführung stoppt, ein Grund außer dem voreingestellten Ereignis ist; der Kern A ferner ausgelegt ist zum: nach dem Empfang einer ersten Stoppabbruchsanweisung und der Wiederaufnahme der Ausführung, Durchführen einer "Debugging"-Informationen-Sammelfunktion zum Sammeln von "Debugging"-Informationen des voreingestellten Ereignisses, und Stoppen der Ausführung nach Beendigung der Durchführung der "Debugging"-Informationen-Sammelfunktion;

der Eingangsport ferner ausgelegt ist zum: Empfangen einer zweiten Stoppabbruchsanweisung und Weiterleiten der zweiten Stoppabbruchsanweisung zum Kern A, wobei die zweite Stoppabbruchsanweisung durch die "Debugging"-Vorrichtung gesendet wird, nachdem die "Debugging"-Vorrichtung einen voreingestellten bevorstehenden Haltepunkt als ein Operationsobjekt des voreingestellten Ereignisses gemäß den "Debugging"-Informationen des voreingestellten Ereignisses einstellt; und der Kern A ferner ausgelegt ist zum: Senden einer Ausführungswiederaufnahmeanweisung zu den anderen Kernen nach dem Empfang der zweiten Stoppabbruchsanweisung und der Wiederaufnahme der Ausführung, wobei die Ausführungswiederaufnahmeanweisung dafür verwendet wird, die anderen Kerne anzuweisen, die Ausführung wiederaufzunehmen, und "Knocking" des bevorstehenden Haltepunktes in einem Prozess der Ausführung des Operationsobjekts des voreingestellten Ereignisses, sodass in einen "Debugging"-Zustand eingetreten wird.

19. Mehrkernprozessor nach Anspruch 18, wobei die "Debugging"-Informationen-Sammelfunktion eine "Debugging"-Schnittstellenfunktion umfasst, eine erste "Debugging"-Ausnahmeanweisung in einem Header der "Debugging"-Schnittstellenfunktion eingestellt ist und die erste "Debugging"-Ausnahmeanweisung dafür verwendet wird, den Kern A anzuweisen, die Ausführung zu stoppen; und der Kern A spezifisch ausgelegt ist zum: nach der Beendigung der Durchführung der Verarbeitungsroutine des voreingestellten Ereignisses, Starten der Durchführung der ersten "Debugging"-Ausnahme-

- anweisung, die sich im Header der "Debugging"-Schnittstellenfunktion befindet, und Senden des Ausführungsstoppsignals zu den anderen Kernen in einem Prozess des Durchführens der ersten "Debugging"-Ausnahmeanweisung.
20. Mehrkernprozessor nach Anspruch 19, wobei die "Debugging"-Informationen-Sammelfunktion ferner eine mit dem voreingestellten Ereignis assoziierte Funktion umfasst, eine zweite "Debugging"-Ausnahmeanweisung ferner in einem "Trailer" der "Debugging"-Schnittstellenfunktion eingestellt ist und die zweite "Debugging"-Ausnahmeanweisung dafür verwendet wird, den Kern A anzuweisen, die Ausführung zu stoppen; und der Kern A spezifisch ausgelegt ist zum: Durchführen der mit dem voreingestellten Ereignis assoziierten Funktion, um die "Debugging"-Informationen des voreingestellten Ereignisses zu sammeln, und dann Durchführen der zweiten "Debugging"-Ausnahmeanweisung, die sich im "Trailer" der "Debugging"-Schnittstellenfunktion befindet.
21. Mehrkernprozessor nach Anspruch 20, wobei sich die mit dem voreingestellten Ereignis assoziierte Funktion in einem Kernel-"Debugging-Stub" befindet und das Kernel-"Debugging-Stub" im Voraus in ein Betriebssystem der Zielmaschine implantiert wird.
22. Mehrkernprozessor nach einem der Ansprüche 18 bis 21, wobei der Mehrkernprozessor in ein System-auf-Chip bzw. SOC integriert ist; der Kern A spezifisch ausgelegt ist zum Senden des Ausführungsstoppsignals zu den anderen Kernen unter Verwendung eines Hardware-"Cross-Trigger"-Netzwerks des SOC, wobei das Ausführungsstoppsignal dafür verwendet wird, die anderen Kerne anzuweisen, die Ausführung zu stoppen; und der Kern A spezifisch ausgelegt ist zum Senden der Ausführungswiederaufnahmeanweisung zu den anderen Kernen unter Verwendung des Hardware-"Cross-Trigger"-Netzwerks des SOC, wobei die Ausführungswiederaufnahmeanweisung dafür verwendet wird, die anderen Kerne anzuweisen, die Ausführung wiederaufzunehmen.
23. "Debugging"-Vorrichtung, wobei die "Debugging"-Vorrichtung auf einer Host-Maschine in einem Fern-"Debugging"-System eingesetzt wird und die "Debugging"-Vorrichtung eine "Debugging"-Einheit und eine Empfangseinheit umfasst, wobei die Empfangseinheit ausgelegt ist zum: Empfangen einer Nachricht, die dafür verwendet wird, einen Kern A anzuweisen, die Ausführung zu stoppen, und Weiterleiten der Nachricht, die dafür verwendet wird, den Kern A anzuweisen, die Ausführung zu stoppen, zu der "Debugging"-Einheit, wobei eine Zielmaschine,
- die sich in dem Fern-"Debugging"-System befindet, einen Mehrkernprozessor umfasst und der Mehrkernprozessor den Kern A umfasst; die "Debugging"-Einheit ausgelegt ist zum: nach der Bestimmung, dass ein Kern A die Ausführung stoppt, Bestimmen eines Grundes dafür, dass der Kern A die Ausführung stoppt; falls der Grund dafür, dass der Kern A die Ausführung stoppt, ein voreingestelltes Ereignis ist, die "Debugging"-Einheit ferner ausgelegt ist zum: Erhalten von "Debugging"-Informationen des voreingestellten Ereignisses und Einstellen eines voreingestellten bevorstehenden Haltepunktes als ein Operationsobjekt des voreingestellten Ereignisses gemäß den "Debugging"-Informationen des voreingestellten Ereignisses, wobei das des voreingestellte Ereignis eine Kernel-Moduscode-Verarbeitungsfunktion oder eine Benutzermoduscode-Verarbeitungsfunktion ist und das Operationsobjekt des voreingestellten Ereignisses ein Kernel-Moduscode oder ein Benutzermoduscode ist; und die "Debugging"-Einheit ferner ausgelegt ist zum Anweisen des Kerns A, die Ausführung wiederaufzunehmen, sodass der Kern A den bevorstehenden Haltepunkt in einem Prozess der Ausführung des Operationsobjekts des voreingestellten Ereignisses "knockt" und der Kern A in einen "Debugging"-Zustand eintritt.
24. Vorrichtung nach Anspruch 23, wobei falls der Grund dafür, dass der Kern A die Ausführung stoppt, ein Grund außer dem voreingestellten Ereignis ist, die "Debugging"-Einheit ferner ausgelegt ist zum Anweisen des Kerns A, die Ausführung wiederaufzunehmen, um die "Debugging"-Informationen des voreingestellten Ereignisses zu sammeln.
25. Vorrichtung nach Anspruch 23 oder 24, wobei die Funktion des Kernel-Moduscodes eine Kernel-Modulladefunktion ist; und die Funktion des Benutzermoduscodes eine Prozesserzeugungsfunktion, eine Prozessschaltfunktion, eine Prozessblockierfunktion, eine Prozessaufweckfunktion, eine Thread-Erzeugungsfunktion, eine Thread-Schaltfunktion, eine Thread-Blockierfunktion oder eine Thread-Aufweckfunktion ist.
26. Vorrichtung nach Anspruch 25, wobei falls das voreingestellte Ereignis die Kernel-Modulladefunktion, die Prozesserzeugungsfunktion oder die Thread-Erzeugungsfunktion ist, die "Debugging"-Informationen des voreingestellten Ereignisses eine Ladeadresse des Operationsobjekts des voreingestellten Ereignisses umfassen; und die "Debugging"-Einheit spezifisch ausgelegt ist zum: Erhalten der Ladeadresse des Operationsobjekts des voreingestellten Ereignisses und Einstellen

des bevorstehenden Haltepunktes als eine Zieladresse des Operationsobjekts des voreingestellten Ereignisses gemäß der Ladeadresse des Operationsobjekts des voreingestellten Ereignisses.

27. Vorrichtung nach Anspruch 25, wobei falls das voreingestellte Ereignis die Prozessschaltfunktion, die Prozessblockierfunktion, die Prozessaufweckfunktion, die Thread-Schaltfunktion, die Thread-Blockierfunktion oder die Thread-Aufweckfunktion ist, die "Debugging"-Informationen des voreingestellten Ereignisses eine Kennung des Operationsobjekts des voreingestellten Ereignisses umfassen; und die "Debugging"-Einheit spezifisch ausgelegt ist zum: Erhalten der Kennung des Operationsobjekts des voreingestellten Ereignisses, Bestimmen einer Ladeadresse des Operationsobjekt des voreingestellten Ereignisses gemäß der Kennung des Operationsobjekts des voreingestellten Ereignisses und Einstellen des bevorstehenden Haltepunktes als eine Zieladresse des Operationsobjekts des voreingestellten Ereignisses gemäß der Ladeadresse des Operationsobjekt des voreingestellten Ereignisses.
28. Vorrichtung nach Anspruch 27, wobei eine Korrespondenz zwischen der Kennung des Operationsobjekt des voreingestellten Ereignisses und der Ladeadresse des Operationsobjekt des voreingestellten Ereignisses in einem Speicherraum der Host-Maschine vorgespeichert wird; und die "Debugging"-Einheit spezifisch ausgelegt ist zum Erhalten, aus dem Speicherraum der Host-Maschine, der Ladeadresse des Operationsobjekt des voreingestellten Ereignisses gemäß der Kennung des Operationsobjekt des voreingestellten Ereignisses und der Korrespondenz.
29. Vorrichtung nach einem der Ansprüche 23 bis 28, wobei die "Debugging"-Einheit einen Debugger und einen "Debugging"-Agenten umfasst; der "Debugging"-Agent spezifisch ausgelegt ist zum: nach der Bestimmung, dass der Kern A die Ausführung stoppt, Bestimmen des Grundes dafür, dass der Kern A die Ausführung stoppt, und, falls der Grund dafür, dass der Kern A die Ausführung stoppt, das voreingestellte Ereignis ist, Melden einer zweiten Nachricht an den Debugger, wobei die zweite Nachricht eine Kennung, die das Stoppen der Ausführung durch den Kern A angibt, und eine Kennung des voreingestellten Ereignisses umfasst; und der Debugger spezifisch ausgelegt ist zum: nach dem Empfang der zweiten Nachricht, Erhalten der "Debugging"-Informationen des voreingestellten Ereignisses gemäß der Kennung des voreingestellten Ereignisses und Einstellen des bevorstehenden Haltepunktes als das Operationsobjekt des voreingestellten Ereignisses gemäß den "Debugging"-Infor-

mationen des voreingestellten Ereignisses; und der Debugger spezifisch ausgelegt ist zum Anweisen des "Debugging"-Agenten, eine zweite Stoppabbruchsanweisung zum Kern A zu senden, wobei die zweite Stoppabbruchsanweisung dafür verwendet wird, den Kern A anzuweisen, die Ausführung wiederaufzunehmen.

30. Vorrichtung nach Anspruch 29, wobei der "Debugging"-Agent spezifisch ausgelegt ist zum: nach der Bestimmung, dass der Kern A die Ausführung stoppt, Bestimmen des Grundes dafür, dass der Kern A die Ausführung stoppt, und, falls der Grund dafür, dass der Kern A die Ausführung stoppt, ein Grund außer dem voreingestellten Ereignis ist, Senden einer ersten Stoppabbruchsanweisung zum Kern A, wobei die erste Stoppabbruchsanweisung dafür verwendet wird, den Kern A anzuweisen, die Ausführung wiederaufzunehmen, um die "Debugging"-Informationen des voreingestellten Ereignisses zu sammeln.
31. Vorrichtung nach einem der Ansprüche 23 bis 29, wobei die "Debugging"-Vorrichtung ferner den Debugger und den "Debugging"-Agenten umfasst; der "Debugging"-Agent spezifisch ausgelegt ist zum: nach der Bestimmung, dass der Kern A die Ausführung stoppt, Bestimmen des Grundes dafür, dass der Kern A die Ausführung stoppt, und, falls der Grund dafür, dass der Kern A die Ausführung stoppt, ein Grund außer dem voreingestellten Ereignis ist, Melden einer ersten Nachricht zum Debugger, wobei die erste Nachricht die Kennung, die das Stoppen der Ausführung durch den Kern A angibt, und eine Standardkennung umfasst; und der Debugger spezifisch ausgelegt ist zum: nach dem Empfang der ersten Nachricht, Anweisen des "Debugging"-Agenten, eine erste Stoppabbruchsanweisung zum Kern A zu senden, wobei die erste Stoppabbruchsanweisung dafür verwendet wird, den Kern A anzuweisen, die Ausführung wiederaufzunehmen, um die "Debugging"-Informationen des voreingestellten Ereignisses zu sammeln.

Revendications

1. Procédé de débogage, appliqué à une machine cible dans un système de débogage distant, dans lequel la machine cible comprend un processeur multicœur, et le procédé comprend :

l'interruption (S101) de fonctionnement, par un cœur A du processeur multicœur après achèvement de l'exécution d'une routine de traitement d'un événement prédéfini, et l'envoi d'un signal d'interruption de fonctionnement à d'autres cœurs dans un processus d'interrup-

- tion de fonctionnement, dans lequel le signal d'interruption de fonctionnement est utilisé pour donner l'instruction aux autres cœurs d'interrompre le fonctionnement, les autres cœurs se réfèrent à tous les cœurs qui sont situés dans le processeur multicœur et qui appartiennent à un même groupe logique que le cœur A, et l'événement prédéfini est une fonction de traitement de code de mode noyau ou une fonction de traitement de code de mode utilisateur ;
- après la réception d'une première instruction de cessation d'interruption et la reprise de fonctionnement, l'exécution (S103), par le cœur A, d'une fonction de collecte d'informations de débogage pour collecter des informations de débogage de l'événement prédéfini, et l'interruption de fonctionnement après achèvement de l'exécution de la fonction de collecte d'informations de débogage, dans lequel la première instruction de cessation d'interruption est envoyée par un appareil de débogage dans le système de débogage distant après que l'appareil de débogage détermine que le cœur A interrompt le fonctionnement et qu'une raison pour laquelle le cœur A interrompt le fonctionnement est une raison autre que l'événement prédéfini ;
- après la réception d'une seconde instruction de cessation d'interruption et la reprise de fonctionnement, l'envoi (S105), par le cœur A, d'une instruction de reprise de fonctionnement aux autres cœurs, dans lequel l'instruction de reprise de fonctionnement est utilisée pour donner l'instruction aux autres cœurs de reprendre le fonctionnement, et la seconde instruction de cessation d'interruption est envoyée par l'appareil de débogage après que l'appareil de débogage définit un point d'arrêt en cours prédéfini en tant qu'objet d'opération de l'événement prédéfini selon les informations de débogage de l'événement prédéfini ; et
- le cognement (S105), par le cœur A, du point d'arrêt en cours dans un processus de fonctionnement de l'objet d'opération de l'événement prédéfini, de façon à entrer dans un état de débogage.
- 2.** Procédé selon la revendication 1, dans lequel la fonction de collecte d'informations de débogage comprend une fonction d'interface de débogage, une première instruction d'exception de débogage est définie dans un entête de la fonction d'interface de débogage, et la première instruction d'exception de débogage est utilisée pour donner l'instruction au cœur A d'interrompre le fonctionnement ; et l'étape consistant à interrompre (S101) le fonctionnement, par le cœur A après achèvement de l'exécution d'une routine de traitement d'un événement prédéfini, et à envoyer un signal d'interruption de fonctionnement à d'autres cœurs dans un processus d'interruption de fonctionnement comprend spécifiquement :
- après achèvement de l'exécution de la routine de traitement de l'événement prédéfini, le fait de commencer, par le cœur A, à exécuter la première instruction d'exception de débogage située dans l'entête de la fonction d'interface de débogage, et l'envoi du signal d'interruption de fonctionnement aux autres cœurs dans un processus d'exécution de la première instruction d'exception de débogage.
- 3.** Procédé selon la revendication 2, dans lequel la fonction de collecte d'informations de débogage comprend en outre une fonction associée à l'événement prédéfini, une seconde instruction d'exception de débogage est en outre définie dans une amorce de fin de la fonction d'interface de débogage, et la seconde instruction d'exception de débogage est utilisée pour donner l'instruction au cœur A d'interrompre le fonctionnement ; et l'étape consistant à exécuter (S103), par le cœur A, une fonction de collecte d'informations de débogage pour collecter des informations de débogage de l'événement prédéfini, et à interrompre le fonctionnement après achèvement de l'exécution de la fonction de collecte d'informations de débogage comprend spécifiquement :
- l'exécution, par le cœur A, de la fonction associée à l'événement prédéfini pour collecter les informations de débogage de l'événement prédéfini, puis l'exécution de la seconde instruction d'exception de débogage située dans l'amorce de fin de la fonction d'interface de débogage.
- 4.** Procédé selon la revendication 3, dans lequel la fonction associée à l'événement prédéfini est située dans un élément de remplacement de débogage de noyau, et l'élément de remplacement de débogage de noyau est implanté à l'avance dans un système d'exploitation de la machine cible.
- 5.** Procédé selon l'une quelconque des revendications 1 à 4, dans lequel la fonction de traitement de code de mode noyau est une fonction de chargement de module de noyau ; et la fonction de traitement de code de mode utilisateur est une fonction de création de processus, une fonction de commutation de processus, une fonction de blocage de processus, une fonction de réveil de processus, une fonction de création de fil, une fonction de commutation de fil, une fonction de blocage de fil, ou une fonction de réveil de fil.
- 6.** Procédé selon la revendication 5, dans lequel si l'événement prédéfini est une fonction de chargement de module de noyau, l'objet d'opération de l'événement prédéfini est un module de noyau ;

- si l'événement prédéfini est la fonction de création de processus, la fonction de commutation de processus, la fonction de blocage de processus, ou la fonction de réveil de processus, l'objet d'opération de l'événement prédéfini est un processus ; et
 si l'événement prédéfini est la fonction de création de fil, la fonction de commutation de fil, la fonction de blocage de fil, ou la fonction de réveil de fil, l'objet d'opération de l'événement prédéfini est un fil.
7. Procédé selon la revendication 5 ou 6, dans lequel les informations de débogage de l'événement prédéfini comprennent un identifiant de l'objet d'opération de l'événement prédéfini ; et
 si l'événement prédéfini est la fonction de chargement de module de noyau, la fonction de création de processus, ou la fonction de création de fil, les informations de débogage de l'événement prédéfini comprennent en outre une adresse de chargement de l'objet d'opération de l'événement prédéfini.
8. Procédé selon l'une quelconque des revendications 1 à 7, dans lequel le processeur multicœur est intégré dans un système sur puce SoC ;
 l'étape d'envoi, par un cœur A, d'un signal d'interruption de fonctionnement à d'autres cœurs, dans lequel le signal d'interruption de fonctionnement est utilisé pour donner l'instruction aux autres cœurs d'interrompre le fonctionnement comprend spécifiquement :
- l'envoi, par le cœur A, du signal d'interruption de fonctionnement aux autres cœurs en utilisant un réseau interdéclenchement matériel du SoC, dans lequel le signal d'interruption de fonctionnement est utilisé pour donner l'instruction aux autres cœurs d'interrompre le fonctionnement ; et
 l'étape d'envoi (S105), par le cœur A, d'une instruction de reprise de fonctionnement aux autres cœurs, dans lequel l'instruction de reprise de fonctionnement est utilisée pour donner l'instruction aux autres cœurs de reprendre le fonctionnement, comprend spécifiquement :
- l'envoi, par le cœur A, de l'instruction de reprise de fonctionnement aux autres cœurs en utilisant le réseau interdéclenchement matériel du SoC, dans lequel l'instruction de reprise de fonctionnement est utilisée pour donner l'instruction aux autres cœurs de reprendre le fonctionnement.
9. Procédé de débogage, exécuté par un appareil de débogage déployé dans une machine hôte, dans lequel la machine hôte est située dans un système de débogage distant, et le procédé comprend :
- la détermination (S102), par l'appareil de débogage après détermination qu'un cœur A interrompt le fonctionnement, d'une raison pour laquelle le cœur A interrompt le fonctionnement, dans lequel le cœur A appartient à un processeur multicœur d'une machine cible dans le système de débogage distant ;
 si la raison pour laquelle le cœur A interrompt le fonctionnement est un événement prédéfini, l'obtention (S104), par l'appareil de débogage, d'informations de débogage de l'événement prédéfini, et le fait de définir un point d'arrêt en cours prédéfini en tant qu'objet d'opération de l'événement prédéfini selon les informations de débogage de l'événement prédéfini, dans lequel l'événement prédéfini est une fonction de traitement de code de mode noyau ou une fonction de traitement de code de mode utilisateur, et l'objet d'opération de l'événement prédéfini est un code de mode noyau ou un code de mode utilisateur ; et
 le fait de donner l'instruction, par l'appareil de débogage, au cœur A de reprendre le fonctionnement, de sorte que le cœur A cogne le point d'arrêt en cours dans un processus de fonctionnement de l'objet d'opération de l'événement prédéfini, et le cœur A entre dans un état de débogage.
10. Procédé selon la revendication 9, dans lequel si la raison pour laquelle le cœur A interrompt le fonctionnement est une raison autre que l'événement prédéfini, le fait de donner l'instruction, par l'appareil de débogage, au cœur A de reprendre le fonctionnement, pour collecter les informations de débogage de l'événement prédéfini.
11. Procédé selon la revendication 9 ou 10, dans lequel la fonction du code de mode noyau est une fonction de chargement de module de noyau ; et
 la fonction du code de mode utilisateur est une fonction de création de processus, une fonction de commutation de processus, une fonction de blocage de processus, une fonction de réveil de processus, une fonction de création de fil, une fonction de commutation de fil, une fonction de blocage de fil, ou une fonction de réveil de fil.
12. Procédé selon la revendication 11, dans lequel si l'événement prédéfini est la fonction de chargement de module de noyau, la fonction de création de processus, ou la fonction de création de fil, les informations de débogage de l'événement prédéfini comprennent une adresse de chargement de l'objet d'opération de l'événement prédéfini ; et
 l'étape consistant à obtenir, par l'appareil de débogage, des informations de débogage de l'événement prédéfini, et à définir un point d'arrêt en cours prédéfini en tant qu'objet d'opération de l'événement prédéfini selon les informations de débogage de

l'évènement prédéfini comprend spécifiquement : l'obtention, par l'appareil de débogage, de l'adresse de chargement de l'objet d'opération de l'évènement prédéfini, et le fait de définir le point d'arrêt en cours prédéfini en tant qu'adresse de destination de l'objet d'opération de l'évènement prédéfini selon l'adresse de chargement de l'objet d'opération de l'évènement prédéfini.

13. Procédé selon la revendication 11, dans lequel si l'évènement prédéfini est la fonction de commutation de processus, la fonction de blocage de processus, la fonction de réveil de processus, la fonction de commutation de fil, la fonction de blocage de fil, ou la fonction de réveil de fil, les informations de débogage de l'évènement prédéfini comprennent un identifiant de l'objet d'opération de l'évènement prédéfini ; et l'étape consistant à obtenir (S104), par l'appareil de débogage, des informations de débogage de l'évènement prédéfini, et à définir un point d'arrêt en cours prédéfini en tant qu'objet d'opération de l'évènement prédéfini selon les informations de débogage de l'évènement prédéfini comprend spécifiquement : l'obtention, par l'appareil de débogage, de l'identifiant de l'objet d'opération de l'évènement prédéfini, la détermination d'une adresse de chargement de l'objet d'opération de l'évènement prédéfini selon l'identifiant de l'objet d'opération de l'évènement prédéfini, et le fait de définir le point d'arrêt en cours prédéfini en tant qu'adresse de destination de l'objet d'opération de l'évènement prédéfini selon l'adresse de chargement de l'objet d'opération de l'évènement prédéfini.
14. Procédé selon la revendication 13, dans lequel une correspondance entre l'identifiant de l'objet d'opération de l'évènement prédéfini et l'adresse de chargement de l'objet d'opération de l'évènement prédéfini est préstockée dans un espace de stockage de la machine hôte ; et l'étape consistant à déterminer une adresse de chargement de l'objet d'opération de l'évènement prédéfini selon l'identifiant de l'objet d'opération de l'évènement prédéfini comprend spécifiquement : l'obtention, à partir de l'espace de stockage de la machine hôte, de l'adresse de chargement de l'objet d'opération de l'évènement prédéfini selon l'identifiant de l'objet d'opération de l'évènement prédéfini et la correspondance.
15. Procédé selon l'une quelconque des revendications 9 à 14, dans lequel l'appareil de débogage comprend un débogueur et un agent de débogage ; l'étape consistant à déterminer, par l'appareil de débogage après avoir déterminé qu'un cœur A interrompt le fonctionnement, une raison pour laquelle le cœur A interrompt le fonctionnement, et si la raison

pour laquelle le cœur A interrompt le fonctionnement est un évènement prédéfini, à obtenir, par l'appareil de débogage, des informations de débogage de l'évènement prédéfini, et à définir un point d'arrêt en cours prédéfini en tant qu'objet d'opération de l'évènement prédéfini selon les informations de débogage de l'évènement prédéfini comprend spécifiquement :

après avoir déterminé, par défilement, que le cœur A interrompt le fonctionnement, le fait de déterminer (S202), par l'agent de débogage, la raison pour laquelle le cœur A interrompt le fonctionnement, et si la raison pour laquelle le cœur A interrompt le fonctionnement est l'évènement prédéfini, le fait de rapporter (S202), par l'agent de débogage, un second message au débogueur, dans lequel le second message comprend un identifiant indiquant que le cœur A interrompt le fonctionnement et un identifiant de l'évènement prédéfini ; et après réception du second message, l'obtention, par le débogueur, des informations de débogage de l'évènement prédéfini selon l'identifiant de l'évènement prédéfini, et le fait de définir le point d'arrêt en cours prédéfini en tant qu'objet d'opération de l'évènement prédéfini selon les informations de débogage de l'évènement prédéfini ; et l'étape consistant à donner l'instruction, par l'appareil de débogage, au cœur A de reprendre le fonctionnement comprend spécifiquement : le fait de donner l'instruction, par le débogueur, à l'agent de débogage d'envoyer une seconde instruction de cessation d'interruption au cœur A, dans lequel la seconde instruction de cessation d'interruption est utilisée pour donner l'instruction au cœur A de reprendre le fonctionnement.

16. Procédé selon la revendication 15, dans lequel l'étape consistant à déterminer, par l'appareil de débogage après avoir déterminé qu'un cœur A interrompt le fonctionnement, une raison pour laquelle le cœur A interrompt le fonctionnement, et si la raison pour laquelle le cœur A interrompt le fonctionnement est une raison autre que l'évènement prédéfini, à donner l'instruction, par l'appareil de débogage, au cœur A de reprendre le fonctionnement, pour collecter les informations de débogage de l'évènement prédéfini comprend spécifiquement : après avoir déterminé, par défilement, que le cœur A interrompt le fonctionnement, le fait de déterminer (S202), par l'agent de débogage, la raison pour laquelle le cœur A interrompt le fonctionnement, et si la raison pour laquelle le cœur A interrompt le fonctionnement est une raison autre que l'évènement prédéfini, l'envoi, par l'agent de débogage, d'une

première instruction de cessation d'interruption au cœur A, dans lequel la première instruction de cessation d'interruption est utilisée pour donner l'instruction au cœur A de reprendre le fonctionnement, pour collecter les informations de débogage de l'évènement prédéfini.

17. Procédé selon l'une quelconque des revendications 9 à 15, dans lequel l'appareil de débogage comprend le débogueur et l'agent de débogage ; et l'étape consistant à déterminer, par l'appareil de débogage après avoir déterminé qu'un cœur A interrompt le fonctionnement, une raison pour laquelle le cœur A interrompt le fonctionnement, et si la raison pour laquelle le cœur A interrompt le fonctionnement est une raison autre que l'évènement prédéfini, à donner l'instruction, par l'appareil de débogage, au cœur A de reprendre le fonctionnement, pour collecter les informations de débogage de l'évènement prédéfini comprend spécifiquement :

après avoir déterminé, par défilement, que le cœur A interrompt le fonctionnement, le fait de déterminer (S202), par l'agent de débogage, la raison pour laquelle le cœur A interrompt le fonctionnement, et si la raison pour laquelle le cœur A interrompt le fonctionnement est une raison autre que l'évènement prédéfini, le fait de rapporter (S202) un premier message au débogueur, dans lequel le premier message comprend l'identifiant indiquant que le cœur A interrompt le fonctionnement et un identifiant par défaut ; et

après réception du premier message, le fait de donner l'instruction, par le débogueur, à l'agent de débogage d'envoyer une première instruction de cessation d'interruption au cœur A, dans lequel la première instruction de cessation d'interruption est utilisée pour donner l'instruction au cœur A de reprendre le fonctionnement, pour collecter les informations de débogage de l'évènement prédéfini.

18. Processeur multicœur, appliqué à une machine cible dans un système de débogage distant, dans lequel le processeur multicœur comprend un cœur A, d'autres cœurs, et un port d'entrée, et les autres cœurs se réfèrent à tous les cœurs qui sont situés dans le processeur multicœur et qui appartiennent à un même groupe logique que le cœur A, dans lequel

le cœur A est configuré pour : interrompre le fonctionnement après achèvement de l'exécution d'une routine de traitement d'un évènement prédéfini, et envoyer un signal d'interruption de fonctionnement aux autres cœurs dans un processus d'interruption de fonctionnement, dans lequel le signal d'interruption de fonctionnement est utilisé pour donner l'instruction

aux autres cœurs d'interrompre le fonctionnement, et l'évènement prédéfini est une fonction de traitement de code de mode noyau ou une fonction de traitement de code de mode utilisateur ;

le port d'entrée est configuré pour : recevoir une première instruction de cessation d'interruption, et réacheminer la première instruction de cessation d'interruption au cœur A, dans lequel la première instruction de cessation d'interruption est envoyée par un appareil de débogage dans le système de débogage distant après que l'appareil de débogage détermine que le cœur A interrompt le fonctionnement et qu'une raison pour laquelle le cœur A interrompt est une raison autre que l'évènement prédéfini ;

le cœur A est en outre configuré pour : après avoir reçu la première instruction de cessation d'interruption et avoir repris le fonctionnement, exécuter une fonction de collecte d'informations de débogage pour collecter des informations de débogage de l'évènement prédéfini, et interrompre le fonctionnement après achèvement de l'exécution de la fonction de collecte d'informations de débogage ;

le port d'entrée est en outre configuré pour : recevoir une seconde instruction de cessation d'interruption, et réacheminer la seconde instruction de cessation d'interruption au cœur A, dans lequel la seconde instruction de cessation d'interruption est envoyée par l'appareil de débogage après que l'appareil de débogage définit un point d'arrêt en cours prédéfini en tant qu'objet d'opération de l'évènement prédéfini selon les informations de débogage de l'évènement prédéfini ; et

le cœur A est en outre configuré pour : envoyer une instruction de reprise de fonctionnement aux autres cœurs après avoir reçu la seconde instruction de cessation d'interruption et avoir repris le fonctionnement, dans lequel l'instruction de reprise de fonctionnement est utilisée pour donner l'instruction aux autres cœurs de reprendre le fonctionnement ; et cogner le point d'arrêt en cours dans un processus de fonctionnement de l'objet d'opération de l'évènement prédéfini, de façon à entrer dans un état de débogage.

19. Processeur multicœur selon la revendication 18, dans lequel

la fonction de collecte d'informations de débogage comprend une fonction d'interface de débogage, une première instruction d'exception de débogage est définie dans un entête de la fonction d'interface de débogage, et la première instruction d'exception de débogage est utilisée pour donner l'instruction au cœur A d'interrompre le fonctionnement ; et

le cœur A est spécifiquement configuré pour : après achèvement de l'exécution de la routine de traitement de l'évènement prédéfini, commencer à exécuter la première instruction d'exception de débogage située dans l'en-tête de la fonction d'interface de

débogage, et envoyer le signal d'interruption de fonctionnement aux autres cœurs dans un processus d'exécution de la première instruction d'exception de débogage.

20. Processeur multicœur selon la revendication 19, dans lequel

la fonction de collecte d'informations de débogage comprend en outre une fonction associée à l'évènement prédéfini, une seconde instruction d'exception de débogage est en outre définie dans une amorce de fin de la fonction d'interface de débogage, et la seconde instruction d'exception de débogage est utilisée pour donner l'instruction au cœur A d'interrompre le fonctionnement ; et

le cœur A est spécifiquement configuré pour : exécuter la fonction associée à l'évènement prédéfini pour collecter les informations de débogage de l'évènement prédéfini, puis exécuter la seconde instruction d'exception de débogage située dans l'amorce de fin de la fonction d'interface de débogage.

21. Processeur multicœur selon la revendication 20, dans lequel la fonction associée à l'évènement prédéfini est située dans un élément de remplacement de débogage de noyau, et l'élément de remplacement de débogage de noyau est implanté à l'avance dans un système d'exploitation de la machine cible.

22. Processeur multicœur selon l'une quelconque des revendications 18 à 21, dans lequel le processeur multicœur est intégré dans un système sur puce SoC ;

le cœur A est spécifiquement configuré pour envoyer le signal d'interruption de fonctionnement aux autres cœurs en utilisant un réseau interdéclenchement matériel du SoC, dans lequel le signal d'interruption de fonctionnement est utilisé pour donner l'instruction aux autres cœurs d'interrompre le fonctionnement ; et

le cœur A est spécifiquement configuré pour envoyer l'instruction de reprise de fonctionnement aux autres cœurs en utilisant le réseau interdéclenchement matériel du SoC, dans lequel l'instruction de reprise de fonctionnement est utilisée pour donner l'instruction aux autres cœurs de reprendre le fonctionnement.

23. Appareil de débogage, dans lequel l'appareil de débogage est déployé sur une machine hôte dans un système de débogage distant, et l'appareil de débogage comprend une unité de débogage et une unité de réception, dans lequel

l'unité de réception est configurée pour : recevoir un message utilisé pour donner l'instruction à un cœur A d'interrompre le fonctionnement et réacheminer le message utilisé pour donner l'instruction au cœur A d'interrompre le fonctionnement à l'unité de débogage, dans lequel une machine cible située dans le

système de débogage distant comprend un processeur multicœur, et le processeur multicœur comprend le cœur A ; l'unité de débogage est configurée pour : après avoir déterminé que le cœur A interrompt le fonctionnement, déterminer une raison pour laquelle le cœur A interrompt le fonctionnement ; si la raison pour laquelle le cœur A interrompt le fonctionnement est un évènement prédéfini, l'unité de débogage est en outre configurée pour : obtenir des informations de débogage de l'évènement prédéfini, et définir un point d'arrêt en cours prédéfini en tant qu'objet d'opération de l'évènement prédéfini selon les informations de débogage de l'évènement prédéfini, dans lequel l'évènement prédéfini est une fonction de traitement de code de mode noyau ou une fonction de traitement de code de mode utilisateur, et l'objet d'opération de l'évènement prédéfini est un code de mode noyau ou un code de mode utilisateur ; et

l'unité de débogage est en outre configurée pour donner l'instruction au cœur A de reprendre le fonctionnement, de sorte que le cœur A cogne le point d'arrêt en cours dans un processus de fonctionnement de l'objet d'opération de l'évènement prédéfini, et le cœur A entre dans un état de débogage.

24. Appareil selon la revendication 23, dans lequel si la raison pour laquelle le cœur A interrompt le fonctionnement est une raison autre que l'évènement prédéfini, l'unité de débogage est en outre configurée pour donner l'instruction au cœur A de reprendre le fonctionnement, pour collecter les informations de débogage de l'évènement prédéfini.

25. Appareil selon la revendication 23 ou 24, dans lequel la fonction du code de mode noyau est une fonction de chargement de module de noyau ; et la fonction du code de mode utilisateur est une fonction de création de processus, une fonction de commutation de processus, une fonction de blocage de processus, une fonction de réveil de processus, une fonction de création de fil, une fonction de commutation de fil, une fonction de blocage de fil, ou une fonction de réveil de fil.

26. Appareil selon la revendication 25, dans lequel si l'évènement prédéfini est la fonction de chargement de module de noyau, la fonction de création de processus, ou la fonction de création de fil, les informations de débogage de l'évènement prédéfini comprennent une adresse de chargement de l'objet d'opération de l'évènement prédéfini ; et l'unité de débogage est spécifiquement configurée pour : obtenir l'adresse de chargement de l'objet d'opération de l'évènement prédéfini, et définir le point d'arrêt en cours prédéfini en tant qu'adresse de destination de l'objet d'opération de l'évènement prédéfini selon l'adresse de chargement de l'objet

d'opération de l'évènement prédéfini.

27. Appareil selon la revendication 25, dans lequel si l'évènement prédéfini est la fonction de commutation de processus, la fonction de blocage de processus, la fonction de réveil de processus, la fonction de commutation de fil, la fonction de blocage de fil, ou la fonction de réveil de fil, les informations de débogage de l'évènement prédéfini comprennent un identifiant de l'objet d'opération de l'évènement prédéfini ; et l'unité de débogage est spécifiquement configurée pour : obtenir l'identifiant de l'objet d'opération de l'évènement prédéfini, déterminer une adresse de chargement de l'objet d'opération de l'évènement prédéfini selon l'identifiant de l'objet d'opération de l'évènement prédéfini, et définir le point d'arrêt en cours prédéfini en tant qu'adresse de destination de l'objet d'opération de l'évènement prédéfini selon l'adresse de chargement de l'objet d'opération de l'évènement prédéfini.
28. Appareil selon la revendication 27, dans lequel une correspondance entre l'identifiant de l'objet d'opération de l'évènement prédéfini et l'adresse de chargement de l'objet d'opération de l'évènement prédéfini est préstockée dans un espace de stockage de la machine hôte ; et l'unité de débogage est spécifiquement configurée pour obtenir, depuis l'espace de stockage de la machine hôte, l'adresse de chargement de l'objet d'opération de l'évènement prédéfini selon l'identifiant de l'objet d'opération de l'évènement prédéfini et la correspondance.
29. Appareil selon l'une quelconque des revendications 23 à 28, dans lequel l'unité de débogage comprend un débogueur et un agent de débogage ; l'agent de débogage est spécifiquement configuré pour : après avoir déterminé que le cœur A interrompt le fonctionnement, déterminer la raison pour laquelle le cœur A interrompt le fonctionnement, et si la raison pour laquelle le cœur A interrompt le fonctionnement est l'évènement prédéfini, rapporter un second message au débogueur, dans lequel le second message comprend un identifiant indiquant que le cœur A interrompt le fonctionnement et un identifiant de l'évènement prédéfini ; le débogueur est spécifiquement configuré pour : après avoir reçu le second message, obtenir les informations de débogage de l'évènement prédéfini selon l'identifiant de l'évènement prédéfini, et définir le point d'arrêt en cours prédéfini en tant qu'objet d'opération de l'évènement prédéfini selon les informations de débogage de l'évènement prédéfini ; et le débogueur est spécifiquement configuré pour donner l'instruction à l'agent de débogage d'envoyer une seconde instruction de cessation d'interruption

au cœur A, dans lequel la seconde instruction de cessation d'interruption est utilisée pour donner l'instruction au cœur A de reprendre le fonctionnement.

30. Appareil selon la revendication 29, dans lequel l'agent de débogage est spécifiquement configuré pour : après avoir déterminé que le cœur A interrompt le fonctionnement, déterminer la raison pour laquelle le cœur A interrompt le fonctionnement, et si la raison pour laquelle le cœur A interrompt le fonctionnement est une raison autre que l'évènement prédéfini, envoyer une première instruction de cessation d'interruption au cœur A, dans lequel la première instruction de cessation d'interruption est utilisée pour donner l'instruction au cœur A de reprendre le fonctionnement, pour collecter les informations de débogage de l'évènement prédéfini.
31. Appareil selon l'une quelconque des revendications 23 à 29, dans lequel l'appareil de débogage comprend en outre le débogueur et l'agent de débogage ; l'agent de débogage est spécifiquement configuré pour : après avoir déterminé que le cœur A interrompt le fonctionnement, déterminer la raison pour laquelle le cœur A interrompt le fonctionnement, et si la raison pour laquelle le cœur A interrompt le fonctionnement est une raison autre que l'évènement prédéfini, rapporter un premier message au débogueur, dans lequel le premier message comprend l'identifiant indiquant que le cœur A interrompt le fonctionnement et un identifiant par défaut ; et le débogueur est spécifiquement configuré pour : après avoir reçu le premier message, donner l'instruction à l'agent de débogage d'envoyer une première instruction de cessation d'interruption au cœur A, dans lequel la première instruction de cessation d'interruption est utilisée pour donner l'instruction au cœur A de reprendre le fonctionnement, pour collecter les informations de débogage de l'évènement prédéfini.

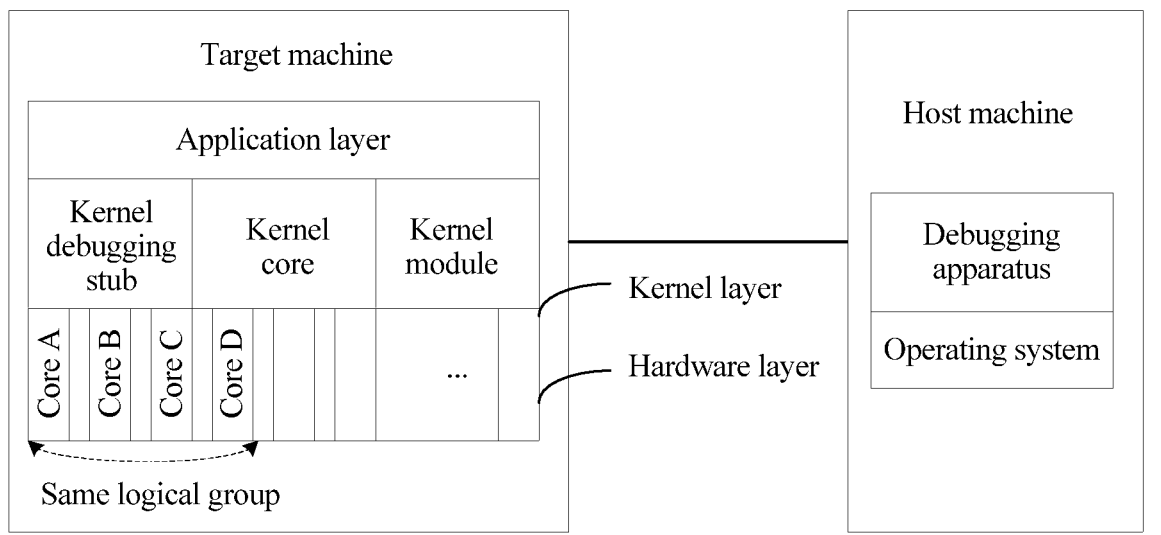


FIG. 1

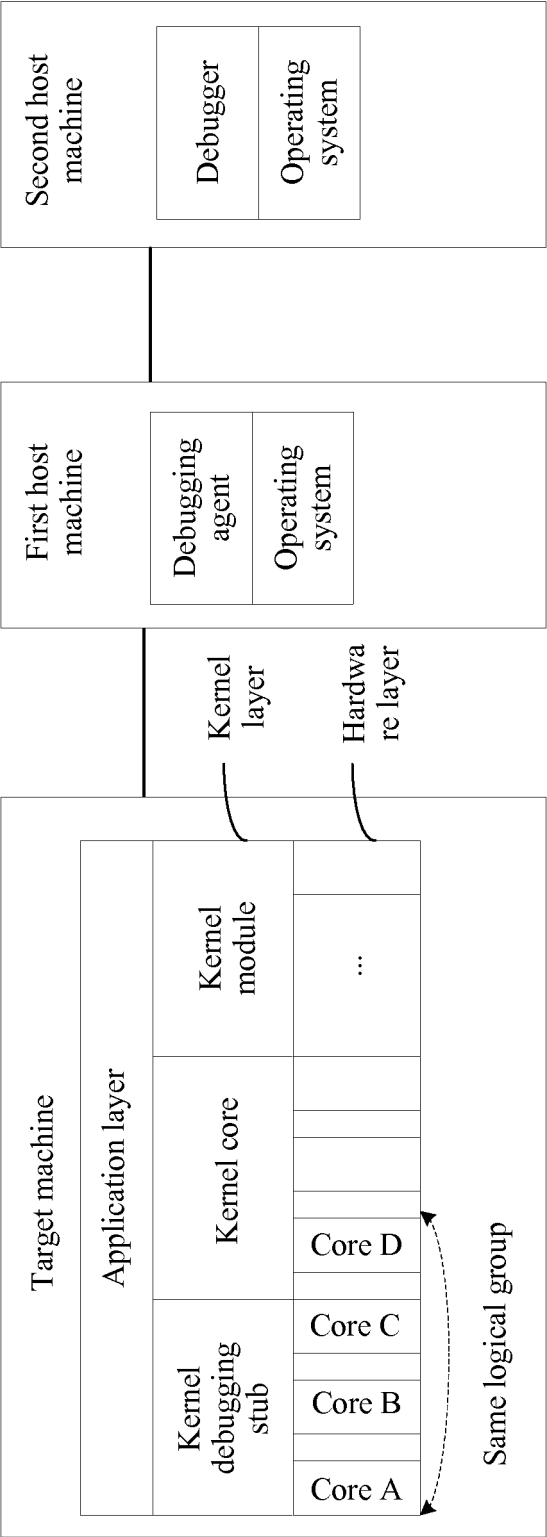


FIG. 2

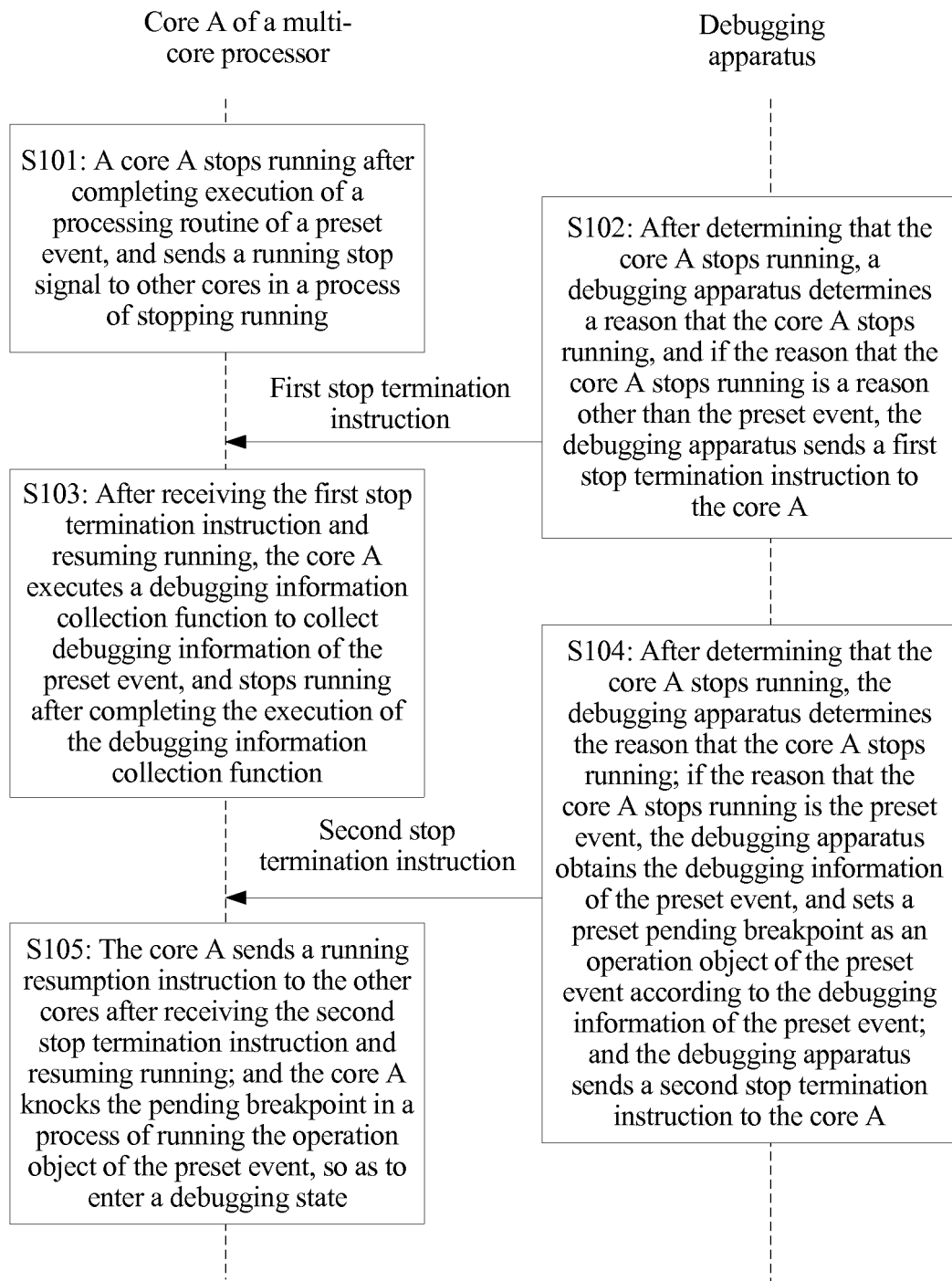


FIG. 3

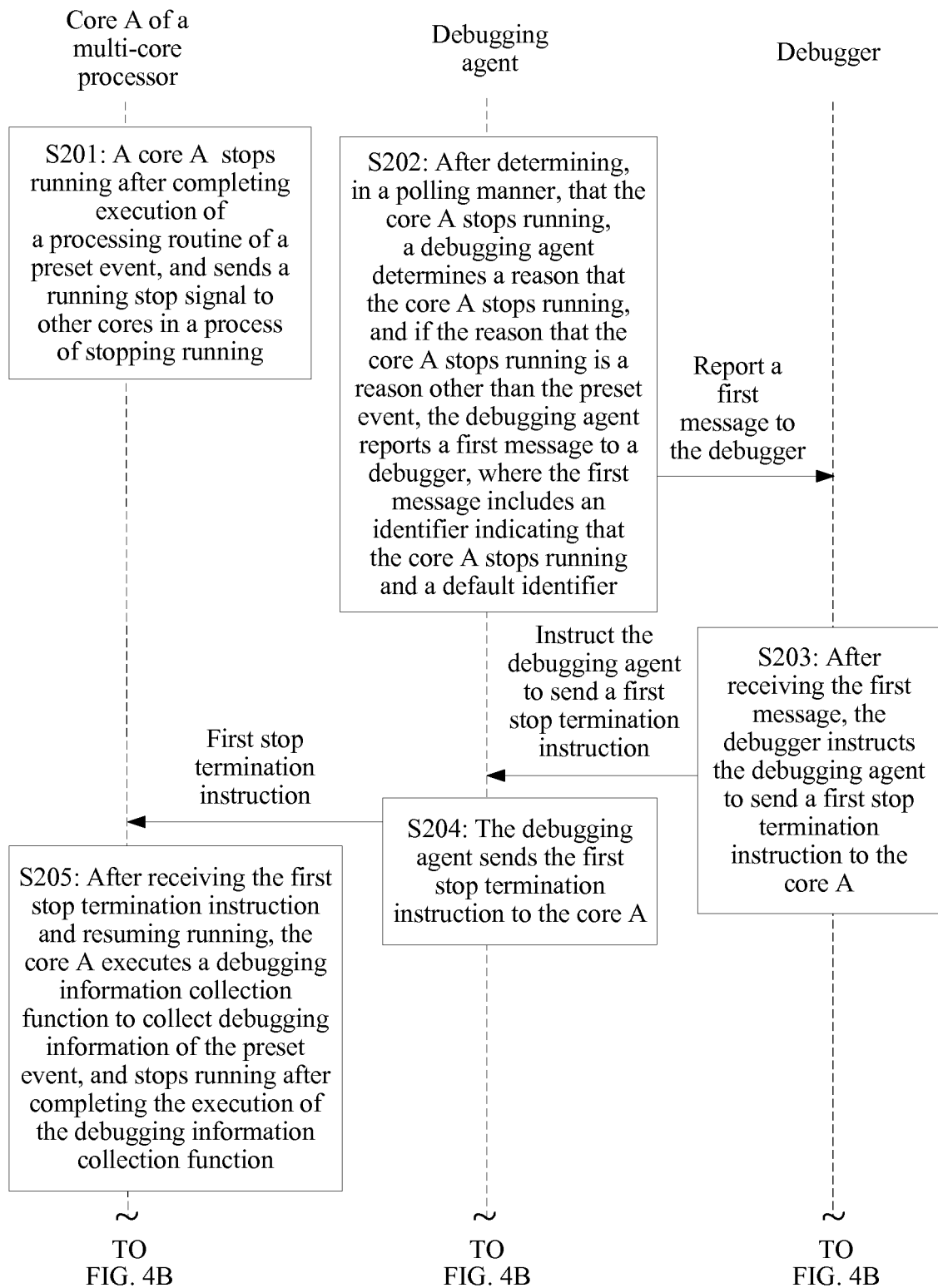


FIG. 4A

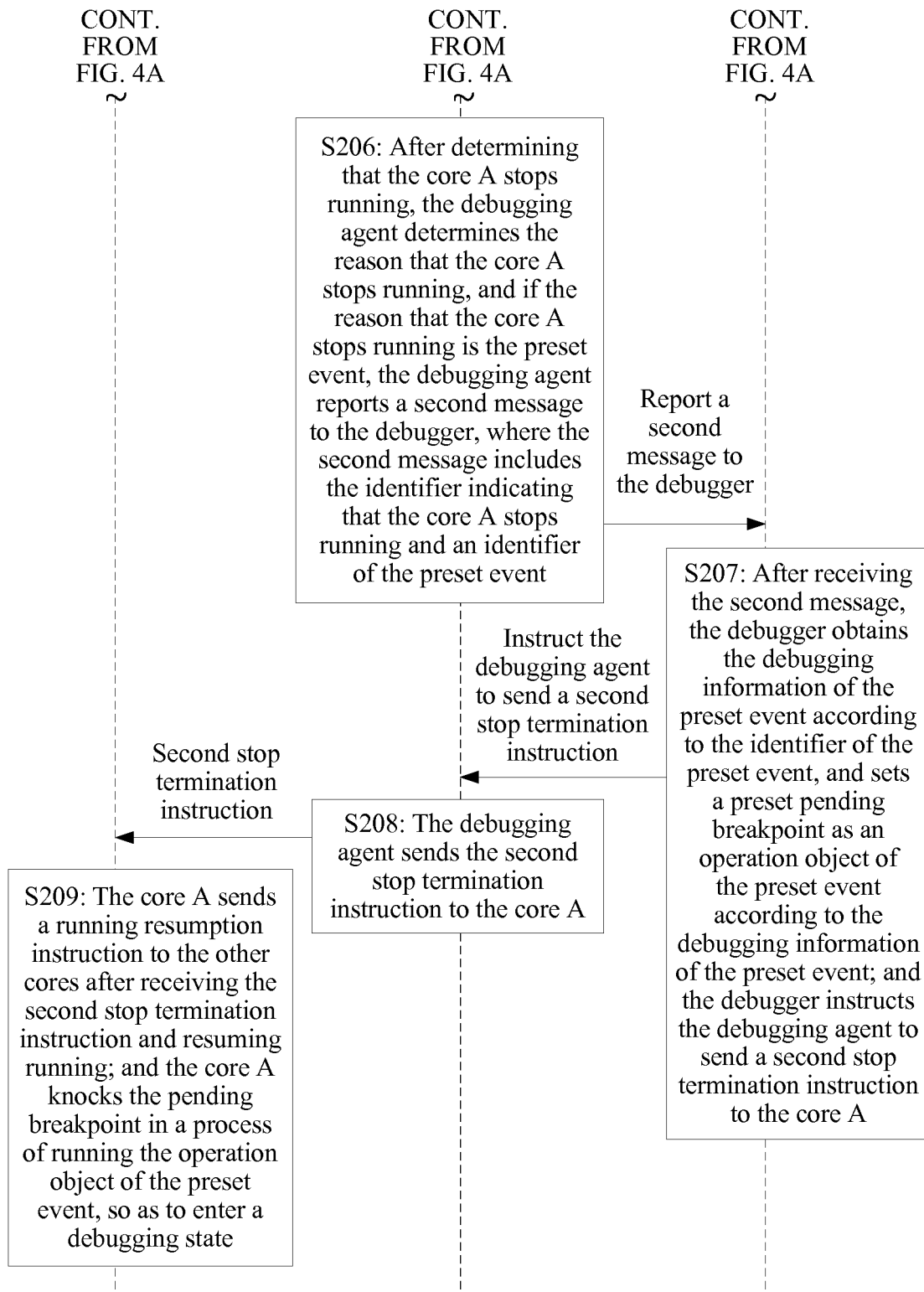


FIG. 4B

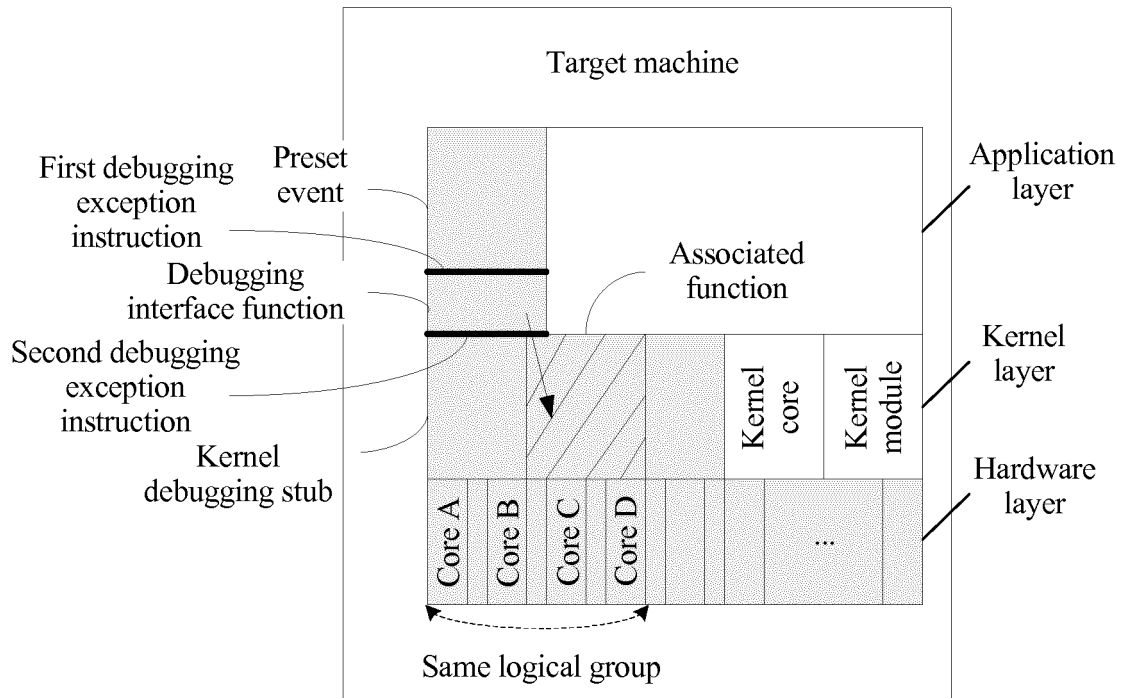


FIG. 5

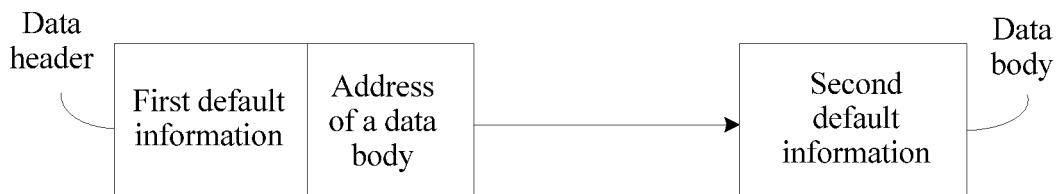


FIG. 6.1

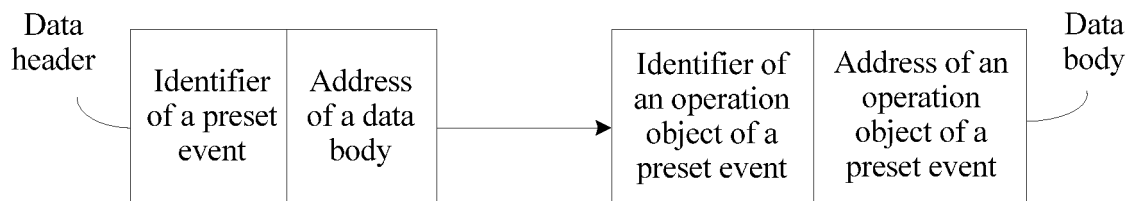


FIG. 6.2

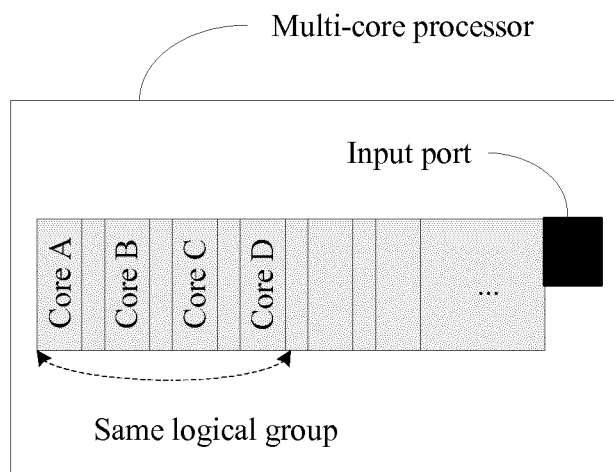


FIG. 7

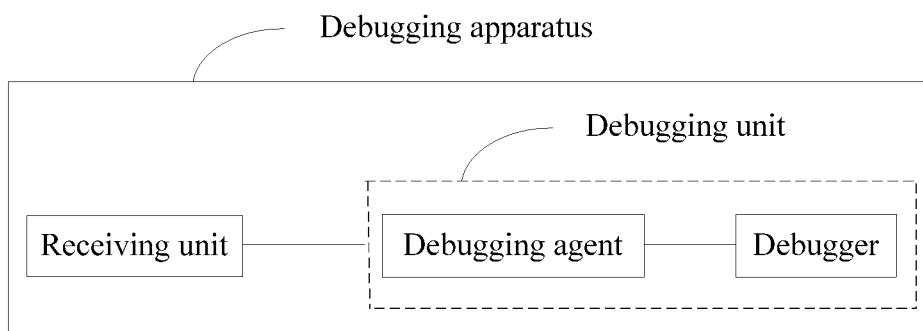


FIG. 8

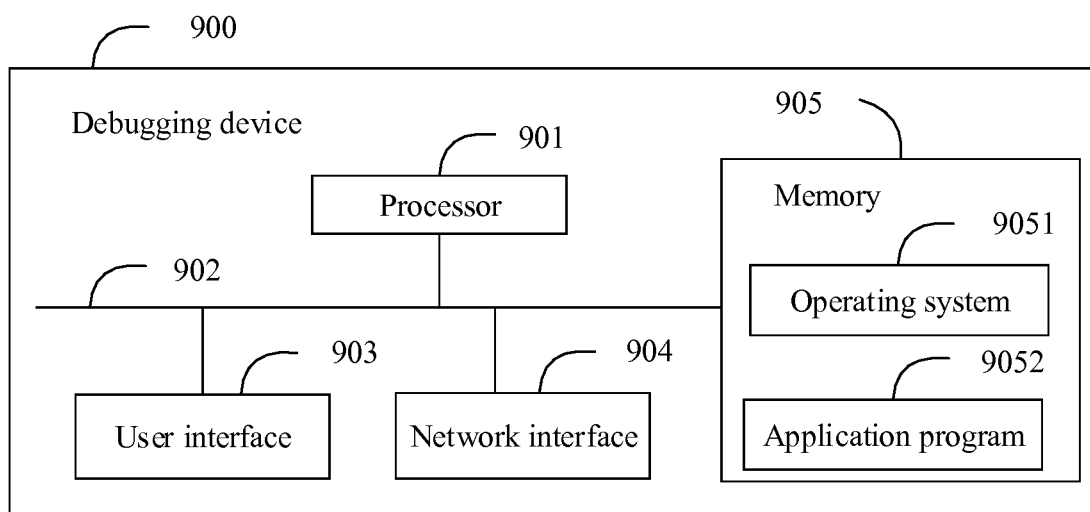


FIG. 9

REFERENCES CITED IN THE DESCRIPTION

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

Patent documents cited in the description

- CN 102346708 A [0004]